

Middlesex University Research Repository

An open access repository of

Middlesex University research

<http://eprints.mdx.ac.uk>

Singh, M. (2010) Use of latent semantic indexing for content based searching and routing of mobile agents on P2P network. Other thesis, Middlesex University. [Thesis]

Final accepted version (with author's formatting)

This version is available at: <https://eprints.mdx.ac.uk/8207/>

Copyright:

Middlesex University Research Repository makes the University's research available electronically.

Copyright and moral rights to this work are retained by the author and/or other copyright owners unless otherwise stated. The work is supplied on the understanding that any use for commercial gain is strictly forbidden. A copy may be downloaded for personal, non-commercial, research or study without prior permission and without charge.

Works, including theses and research projects, may not be reproduced in any format or medium, or extensive quotations taken from them, or their content changed in any way, without first obtaining permission in writing from the copyright holder(s). They may not be sold or exploited commercially in any format or medium without the prior written permission of the copyright holder(s).

Full bibliographic details must be given when referring to, or quoting from full items including the author's name, the title of the work, publication details where relevant (place, publisher, date), pagination, and for theses or dissertations the awarding institution, the degree type awarded, and the date of the award.

If you believe that any material held in the repository infringes copyright law, please contact the Repository Team at Middlesex University via the following email address:

eprints@mdx.ac.uk

The item will be removed from the repository while any claim is being investigated.

See also repository copyright: re-use policy: <http://eprints.mdx.ac.uk/policies.html#copy>

Use of Latent Semantic Indexing for Content Based Searching and Routing of Mobile Agents on P2P Network

A thesis submitted to Middlesex University
in partial fulfilment for the degree of Master of Philosophy

by M. Singh

School of Engineering and Information Sciences

Middlesex University

March 2010

Abstract

The peer-to-peer (P2P) system has a number of nodes that are connected to each other in an unstructured or a structured overlay network. One of the most important problems in a P2P system is locating of resources that are shared by various nodes. Techniques such as Flooding and Distributed Hash-Table (DHT) has been proposed to locate resources shared by various nodes. Flooding suffers from saturation as number of nodes increase, while DHT cannot handle multiple keys to define and search a resource. Various further research works including multi agent systems (MAS) have been pursued that take unstructured or structured networks as a backbone and hence inherently suffer from problems. We present the solution that is more efficient and effective for discovering shared resources on a network that is influenced by content shared by nodes. Our solution presents use of multiple agents that manage the shared information on a node and a mobile agent called Reconnaissance Agent (RA), that is responsible for querying various nodes. To reduce the search load on nodes that have unrelated content, an efficient migration route is proposed for RA, that is based on cosine similarity of content shared by nodes and user query. Results show reduction in search load and traffic due to communication, and increase in recall value for locating of resources defined by multiple keys using RA that are logically similar to user query. Furthermore, the results indicate that by use of our technique the relevance of search results is higher; that is obtained by minimal traffic generation/communication and hops made by RA.

Keywords: Resource Discovery, P2P, Reconnaissance Agent, Latent Semantic Indexing, Cosine Similarity.

Acknowledgements

I would like to acknowledge and extend my heartfelt gratitude to the following persons who have made the completion of this thesis possible:

My supervisors, Dr. Xiaochun Cheng and Dr. Roman Belavkin, for their vital assistance, encouragement, and support throughout the duration of the research project.

Late Emeritus Prof. Colin Tully for giving me the chance to pursue postgraduate studies. His initial encouraging words have been driving force for me throughout this work. May he rest in peace.

Mr. Sumeet Gautam for his assistance in collection of the topics for background chapter and testing.

Most especially to my family

And to **Akal Purakh Waheguru**, who made all things possible.

M. Singh

Middlesex University

Contents

List of Figures	vii
List of Tables	ix
1 Introduction	1
1.1 Motivation & Background	1
1.2 Research Question	3
1.3 Aims and Objectives	3
1.4 Research Method	4
1.5 Contributions	4
1.6 Structure of Report	5
2 Literature Survey	6
2.1 Indexing Architectures used by P2P Systems	6
2.1.1 <i>Centralised Indexing</i>	
2.1.2 <i>Decentralised Indexing - Unstructured Network</i>	
2.1.3 <i>Distributed Indexing - Structured Network</i>	
2.2 Resource Discovery and Routing	12
2.2.1 <i>Resource Discovery in Unstructured P2P Systems</i>	
2.2.2 <i>Resource Discovery in Structured P2P Systems</i>	
2.2.3 <i>Resource Discovery in Mobile Agent Systems</i>	
2.3 Critical Review	20
2.4 Agent Based System Development Frameworks	20
2.5 Qualitative Comparison of Mobile Agent Platforms	21
2.6 Summary of the Chapter	27
3 Design Features and Implementation	28
3.1 The Proposed Multi-Agent System for Resource Discovery - Affinity	28
3.1.1 <i>The Proposed Global System Architecture</i>	
3.1.2 <i>Specification of Agents</i>	
3.2 The Proposed Mobile Agent Routing	32
3.2.1 <i>Latent Semantic Indexing and Singular Value Decomposition (LSI-SVD) for Peer Clustering and Mobile Agent Routing</i>	
3.3 The Proposed Multi-Agent Collaboration for Resource Discovery	36
3.4 Implementation	39

3.4.1	<i>Agents Communication Implementation</i>	
3.4.2	<i>Feature Matrix - Frequency-Based Indexing</i>	
3.4.3	<i>Implementation of Latent Semantic Indexing and Singular Value Decomposition</i>	
3.4.3.1	<i>Index Maintenance</i>	
3.4.4	<i>Similarity Function</i>	
3.4.4.1	<i>Node Learning - Clustering</i>	
3.4.4.2	<i>Node Searching and Ranking - Content Based Routing</i>	
3.4.4.3	<i>Query Resolving</i>	
3.4.5	<i>Mobile Agent - Reconnaissance Agent</i>	
3.5	Discussion	54
3.6	Summary of the Chapter	54
4	Experiments, Results, and Evaluation	56
4.1	Design of Experiments	56
4.1.1	<i>Experiment Environment and Test Bed</i>	
4.2	Test 1 - Comparison to Flooding Technique	60
4.2.1	<i>Experiment 1 - Response Time and Evaluation</i>	
4.2.2	<i>Experiment 2 - Effectiveness of Search Technique and Evaluation</i>	
4.2.3	<i>Observations</i>	
4.2.4	<i>Critical Analysis</i>	
4.3	Test 2 - Comparison to Other Routing Techniques/Algorithms	65
4.3.1	<i>Experiment 1 - Pair-Wise Document Similarity And Evaluation</i>	
4.3.2	<i>Experiment 2 - Effectiveness of Search Technique And Evaluation</i>	
4.3.3	<i>Experiment 3 - Effectiveness to Locate Resources and Evaluation</i>	
4.3.4	<i>Experiment 4 - Degree of Relevance of Results and Evaluation</i>	
4.4	Discussion	71
4.5	Summary of the Chapter	72
5	Discussion, Conclusions, and Future Work	73
5.1	Discussions - Analysis of Other Research Works	73
5.2	Applications for Research Conducted	76
5.3	Conclusions	77
5.4	Future Work	78
	References	80
A	Similarity Measures and Weighting Functions	87
B	Classes Realised - Affinity	88
C	Program Listings - Affinity	89
C.1	Interface BootInf.java	89
C.2	Class Bootstrap.java	90
C.3	Class BootstrapServer.java	95
C.4	Class Extractor.java	97

C.5	Class ReportSim.java	99
C.6	Class Directory.java	101
C.7	Class MasterList.java	102
C.8	Class Repository.java	110
C.9	Class Node.java	112
C.10	Class InformationAgent.java	115
C.11	Class LocalAgent.java	130
C.12	Class LocalUI.java	146
C.13	Class MatchStore.java	148
C.14	Class InterfaceAgent.java	149
C.15	Class SearchGUI.java	154
C.16	Class ReconnaissanceAgent.java	160

List of Figures

2.1	A typical scenario of the centralised system. Source: Singh et al. (2009) . . .	7
2.2	Illustration of the flooding process.	8
2.3	Comparison of distributed indexing structures. (i) Gnutella-like local indexing. (ii) Global indexing. (iii) Hybrid indexing. (iv) Optimized hybrid indexing. a, b, and c are terms. X, Y, and Z are documents. Source: Tang & Dwarkadas (2004)	17
2.4	The SMART architecture Source: Wong et al. (2001)	22
2.5	The D'Agent architecture	23
2.6	The simplified version of Grasshopper Architecture. The basic services include MASIF and Core Services. MASIF includes agent creation, destruction, suspend, activate and location services and Core services include agent execution, transport, management, communication, security and naming. The enhanced services include APIs, GUIs and task control features.	24
2.7	The Aglets architecture Source: Schoeman & Cloete (2003)	25
3.1	Global architecture of the system	30
3.2	Peer clustering and overlay organisation achieved using <i>latent semantic indexing</i>	33
3.3	Interactions between multiple agents for resource discovery and realisation of an overlay network	37
3.4	The RA's interaction with InfA for issuing new peer GUID	39
3.5	Flow diagram for behaviour of the InfA and the LA upon arrival of the RA	40
3.6	<i>ACLMessage</i> from the RA to the LA for search request. <i>ACLMessage</i> received by the LA from the RA using <i>MessageTemplate</i> and replying with <i>setContentObject</i> or in blocked state.	41
3.7	Creation of the RA in the method <i>onGUIEvent()</i> from class <i>InterfaceAgent</i>	43
3.8	<i>ReceiveMessageRecon</i> class showing <i>blocked state</i> of when reply received is <i>null</i> and the <i>MessageTemplate</i> for receiving messages from the RA	43
3.9	Method <i>getKeywords()</i> for getting keywords and their frequencies and holding in data structure <i>TreeMap</i>	45
3.10	Realisation of Singular Value Decomposition from frequency based keyword-resource or keyword node matrix	46
3.11	Index Maintenance task performed recursively by Information Agent	47
3.12	Directory data structure used by <i>BootStrapServer</i> to pass clustered nodes result to <i>InformationAgent</i>	49

3.13	Realisation of node searching and ranking	51
4.1	Frequency distribution of response time analysis - Gnutella vs. Affinity	61
4.2	Query successful vs. unsuccessful - Affinity method	62
4.3	Query successful vs. unsuccessful - flooding method	62
4.4	Division of packets for Gnutella	63
4.5	Gnutella packets analysed using Wireshark	64
4.6	Precision and recall results comparing LSI to TF-IDF indexing model	65
4.7	Pair-wise document similarity TF-IDF Jaccard vs. LSI Cosine	66
4.8	Number of times a document appears for 30 queries Jaccard similarity vs. Cosine similarity	68
4.9	Number of documents found for 30 separate queries on corpus of documents .	69
4.10	Similarity score distribution TF-IDF Jaccard vs. LSI Cosine	70
B.1	Classes for resource discovery system - Affinity	88

List of Tables

2.1	A classification of P2P routing infrastructures in terms of network structures Source: Androutsellis-Theotokis & Spinellis (2004)	11
2.2	Summary of infrastructure for routing and resource discovery location Source: Androutsellis-Theotokis & Spinellis (2004)	12
2.3	Comparison of features of routing algorithms Source: Prakash (2006)	13
2.4	Qualitative Comparison Among Mobile Agent Platforms Source: Trillo et al. (2007)	25
4.1	Gnutella flooding peers test bed	58
4.2	MAS test bed	59
4.3	Keywords used for sharing resource on each node	59
A.1	Similarity measures	87
A.2	Local and global weighting functions	87

CHAPTER 1

INTRODUCTION

1.1 Motivation & Background

The volume of data published online per year is estimated to be of an order of approximately one terabyte and it is expected to grow exponentially. The solution offered to users is in form of a search engine, for instance Google. However, these solutions suffer from requirement of maintaining a large centralised database about online published information. In order to support the solution and also offer scalability, they require a large and highly costly hierarchical infrastructure. Moreover, any newly published information requires time for indexing and is often not indexed for weeks. Similarly, any information that has either been removed or ceases to exist also results in dead-links for users because of delayed indexing.

These reasons call for a requirement of a scalable infrastructure that is capable of indexing, routing and searching rich published content.

As opposed to centralised form for indexing offered by search engines, peer-to-peer (P2P) networks offer solution for resource discovery by making the task of hosting distributed. The P2P networks consist of a number of decentralised nodes sharing their resources on an overlay network. Here the resources mean services/files that are hosted on nodes of the network. P2P systems offer low-cost sharing of information and with high autonomy. P2P networks offer characteristics such as high availability, low cost and ease of deployment, data freshness and good scalability Yingwu Zhu (2005). Because of following features P2P networks become ideal choice as opposed to centralised solutions offered by search engines.

1. Autonomy: Autonomy of nodes allows them to join/leave at any time, control their data with respect to other nodes i.e. shared resources are published and indexed

immediately.

2. Query expressiveness: Key-lookup, key-word search
3. Efficiency: Efficient use of bandwidth, computing power and storage

However, the process of discovery of this shared information is not very efficient due to poor search performance and unavailability of heuristics Tran & Schonwalden (2008).

A classical client and server based centralised solution to a location of resource is offered by Napster Napster (2003); Aberer et al. (2004). In this approach, a client connects to central server - that is responsible for indexing resources and their location. Upon query about resource location from any other client, the central server issues the IP address of the client where resource is located. This solution cripples autonomy of a client due to centralised sever, as in case of server failure, clients cannot locate resources.

Another approach to resource location is offered by Gnutella, where the decentralised peers communicate to other peers when the resource location query is issued by user Chawathe et al. (2003); Forum (2002). This solution offer high degree of autonomy as peers can join or leave the overlay network without affecting rest of the network. When locating a resource, peer floods the user query on the overlay network usually with time-to-live constraint in order to query other peers about required resource. The inefficiency in this approach attributed to three facts:

1. The overlay network is created randomly as there is not structure associated with it
2. The queries for a resource location are forwarded “blindly” from one peer to another peer using technique called flooding due to which there is unnecessary quantity of message on the network
3. Saturation as number of nodes increase.

A more “rigid” approach is taken by a structured overlay that is based on hash functions supports key-based routing such that resource identifiers are mapped to the peer identifier address space and a resource request is routed to the nearest peer in the peer address space Ratnasamy et al. (2001); Rowstron & Druschel (2001); Stoica et al. (2001); Zhao et al. (2001). Although such systems are better than unstructured overlay from performance point of view as some heuristics are available for locating a resource (only where the search

keys are known exactly), but they are not as effective for approximate keywords, or text based resource location Yingwu Zhu (2005); Tran & Schonwalden (2008).

1.2 Research Question

The author formulates the overall research question as following:

Can the process of resource discovery be improved for P2P systems in order to increase search performance such that the higher number of relevant results can be achieved and keep the possibility of saturation of network low that is a resultant of routing on P2P network?

It is understood from literature that saturation can be decreased and hence improved, if informed search is performed that is resultant of availability of heuristics and that the search performance or recall can be increased if an efficient indexing technique and similarity functions are available. This results into breaking down of general research questions into:

1. Can global heuristics be distributed to nodes on the overlay network efficiently with constraint on communication overhead?
2. How can search performance or recall and routing be improved dynamically?
3. What type of characteristics and representation must the resource have in order to be indexed and further be used for representing the node?

1.3 Aims and Objectives

The main aim of this research is to design and implement a novel routing and searching technique based on Latent Semantic Indexing (LSI) and mobile agent technology in a P2P network created by collaboration of multiple agents.

The main objective is to design and implement a resource discovery system that uses mobile agent technology for discovering and selecting nodes and for routing the mobile agent through overlay network based on content of query with purpose of minimising response time, reducing possible delays, maximising network performance by reducing the possibility of saturation and maximising the recall by providing relevant results. Furthermore, it is endeavoured that this system will offer improvement over attributes of performance and scalability.

1.4 Research Method

The field of using mobile agents on P2P networks using LSI is fairly new and most previous attempts have been made using term-based matching techniques, flooding, Distributed Hash Table (DHT) on unstructured or structured networks. It can be concluded without a doubt that this field is growing rapidly and is not very well understood at this stage. The author concludes that the most suitable research method for this research project is experimental research where the evaluation of various experiments conducted will be compared both quantitatively and qualitatively to other related works in this field.

The author endeavours to conduct experiments in order to answer the research question and prove the postulated hypothesis that mobile agent can be used effectively for efficient resource discovery when powered by content-based routing to create network heuristics and discover the topology of overlay network for the purpose of maximising search performance, minimising response time, have higher inter-cluster links and higher degree of relevance of the obtained search results.

1.5 Contributions

The purpose of this research is to offer the multi-agent system (MAS) and the resource discovery based on content based routing of mobile agent that overcomes the disadvantages of structured overlay i.e. be able to locate resources even when the keys are unknown, approximate, or text based multiple keys and also offer the flexibility characteristic of autonomous unstructured overlay but by reducing number of message on the network and control or remove unnecessary flooding.

Through this research work, the author proposes the following:

- *Autonomous MAS System*: a flexible multi-agent based approach for dynamic organisation of P2P network that is based on the similarity of content shared by peers. The similarity of content between two or more peers is translated into similarity between peers or a cluster of peers sharing similar content.
- *Deterministic Content Driven Routing*: the resource location mechanism that uses semantic similarity between content shared by peers and search keywords to deterministically route a mobile agent called the reconnaissance agent (RA) to peers that host content that is similar to a user query.

- *Use LSI Based Indexing and Query Matching*: the use of LSI and cosine similarity by RA to find relevance of resource(s) hosted by peer as a best match for a user query (where the user query can be text based or an approximate query).

The author demonstrates that this method improves the resource discovery performance i.e. finding a relevant resource(s) with lower response time and hence reducing search load.

1.6 Structure of Report

The rest of the report is structured as follows.

Chapter 2 surveys the current literature and draws lessons to propose the capabilities that a resource discovery system should obtain. In doing so, chapter 2 also collates a large amount of research work relevant to field of study and also discusses the architecture and platforms for development of MAS.

Chapter 3 describes the design features of proposed MAS based resource discovery system, node clustering based on semantic similarity of content hosted by nodes and RA routing, and the multi-agent collaboration for resource discovery. Furthermore, it describes the implementation done using Java remote method invocation (RMI) and Java Agent Development Framework (JADE) Bellifemine et al. (2007).

Chapter 4 is dedicated for experimentation where the effectiveness of proposed resource discovery algorithm and resource locating algorithm is compared against flooding (Gnutella) in terms of response time and search load. Furthermore, proposed node clustering algorithm for routing the RA on the overlay network, messages on network and relevance of results obtained due to user invoked query is compared to contemporary research work done by other researchers in field of using mobile agents for resource discovery.

Chapter 5 is dedicated for discussions for assembling and comparing our concepts to other related works in the field of resource discovery and further provide list the conclusions and also presents the future work that can be conducted in this field.

Ending sections of report provide references, appendices, and program listings.

CHAPTER 2

LITERATURE SURVEY

This chapter provides a detailed survey of current literature and draw lessons to propose the capabilities that a resource discovery system should obtain. It also collates a large amount of research work relevant to field of study and also discusses the architecture and platforms for development of MAS.

As described by Singh et al., there are diverse set of solutions that are available for resource discovery. These solutions are characterised through the routing strategy and resource searching strategy that is applied by them Karnstedt et al. (2004); Singh et al. (2009). The author have categorised and reviewed the resource searching techniques used by unstructured and structured P2P systems by initially discussing architectures. The author also presents most current search techniques that are being introduced to the resource discovery domain.

2.1 Indexing Architectures used by P2P Systems

2.1.1 Centralised Indexing

The first most popular P2P Network was Napster, which used Central Indexing Server for storing the locations of the resources Aberer et al. (2004). Using this network Napster client's in the network can communicate with the other Napster clients. In Napster a dedicated central server maintains an index of the files shared by the active peers on the network. Each peer in the network maintains a constant connection to one of the central server through which the query for file location is sent. When a central indexing server receives the query for a file location it cooperates to process the query and returns the corresponding matching file locations to the peer making the query. After the peer making

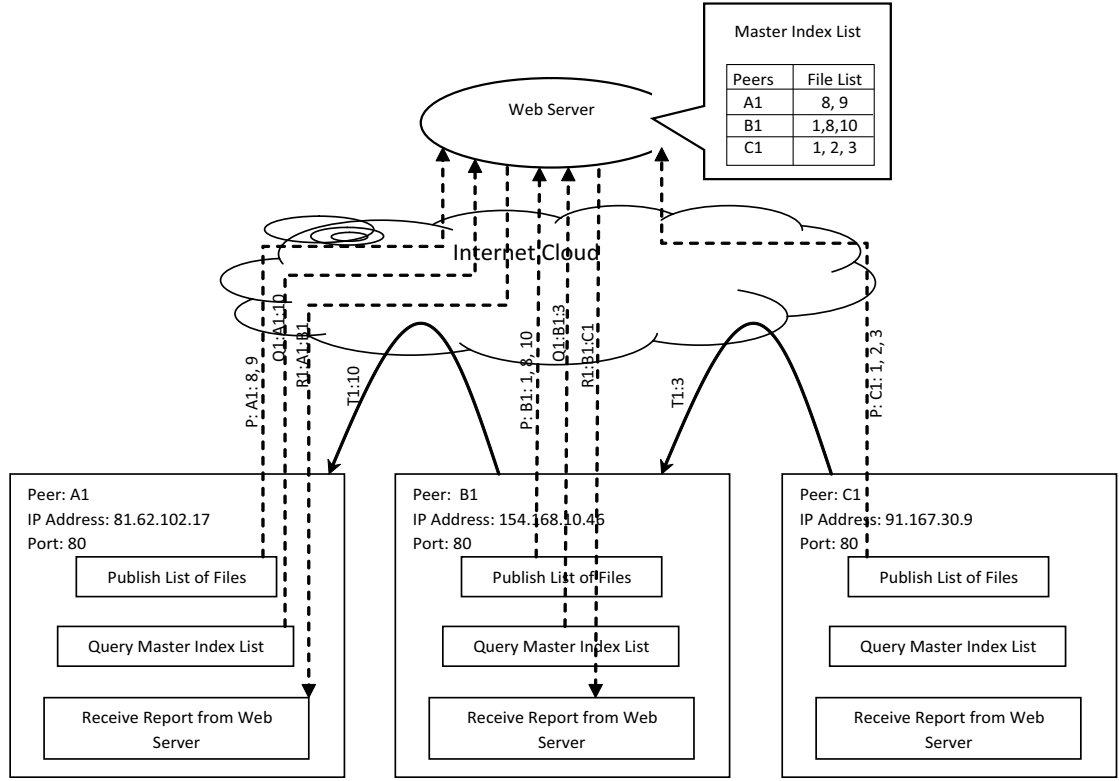


FIGURE 2.1: A typical scenario of the centralised system. Source: Singh et al. (2009)

query receives response from the indexing server about the list of locations of the resource, the peer can now make direct communication with the peers having the resources and initiate the transfer of the resource. Besides maintaining the list of resources in the network, the indexing server also keeps track of each peer that is active or monitors the state of the peer like keeping track of the information of the peer for instance the duration the peer has been active or the connection speed the peer is at Napster (2003). In Figure 2.1, the peer A1, peer B1 and peer C1 are sharing resources 8, 9; 1, 8, 10 and 1, 2, 3 respectively. The central server, “Napster.com” that keeps the index of all resources shared by the peers. The central server is queried by the peer A1 and peer B2 for the resource 10 and resource 3 respectively. The central server replies by providing the IP address of the resource providers to each of the peers. The direct connection is established between two peers for downloading of the resource.

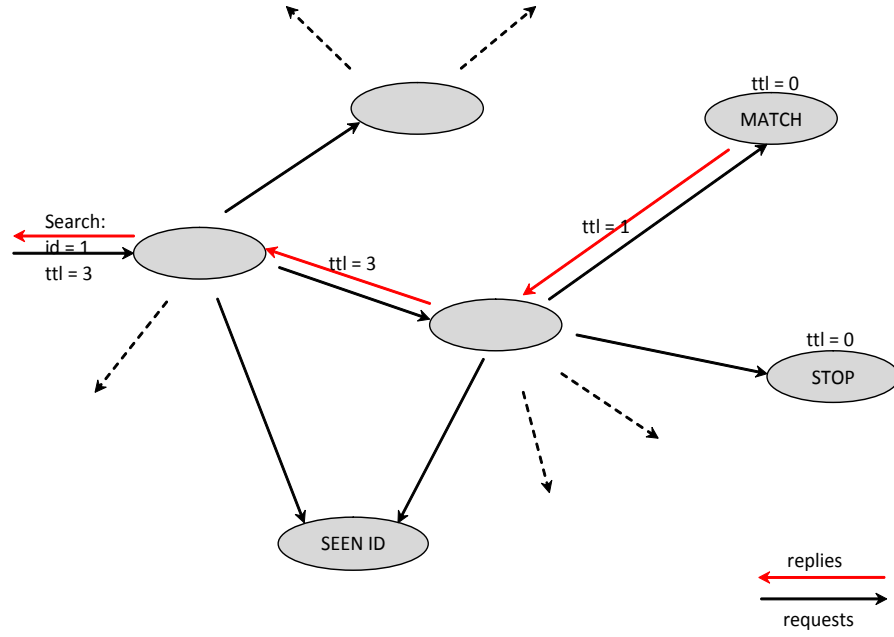


FIGURE 2.2: Illustration of the flooding process.

2.1.2 Decentralised Indexing - Unstructured Network

An unstructured overlay like GNUTELLA is organised into random graph topology where there is no specific topology that the overlay network follows and it uses flooding or random walks to discover resource in the network. This overlay is constructed easily when a node wants to join the network. During the resource discovery each node visited will evaluate the query locally on its data store. Before starting to exchange messages between the nodes, a Gnutella node connects itself to the network by connecting with another *well-known* node on the network. Once the connection is established, the addresses of one or more host will be supplied as the node joins the network. The listening node is advertised by Pong messages. When another node is located on the network TCP/IP connection is established and a handshake sequence is initiated. In Figure 2.2, it is observed that when the search begins from id=1, it is broadcasted to all the peers that are connected to the node with TTL=3. The TTL is decreased by 1 after every hop until TTL drops to zero. If the matching resource is found it is responding through the reverse path until it reaches to the originating node id=1. Details of Gnutella resource discovery protocol are discussed in Section 2.2.1.

2.1.3 Distributed Indexing - Structured Network

A structured overlay and DHT based systems like Chord, Pastry, Content Addressable Network (CAN), and Tapestry are the improvement on unstructured overlay to improve the performance of resource discovery Stoica et al. (2001); Rowstron & Druschel (2001); Ratnasamy et al. (2001); Zhao et al. (2004). It ensures that any node can efficiently route a search to some peer that has the desired file even in the rare availability Killmeyer (2006). The nodes in the network impose constraints on the topology as well as on the data placement to provide with efficient search mechanism and resource discovery. In all the DHT systems mentioned above files are associated with a key and each node in the network is responsible for storing list of resources hence having list of keys. The first and foremost operation in the DHT system is the look up for the key as $\text{lookup}(\text{key})$ which returns a location of the resource or the key and hence IP address.

Chord

Till date there are many load balancing approaches, Chord was the first to propose the concept of virtual servers and hence address the load balancing by having each node simulate a logarithmic number of virtual servers Zhu & Hu (2005). Using Chord, only $\log(N)$ messages are required to find the resource in the Chord Network where N being the number of active nodes in the network. Chord allows distributed nodes to agree on a single Chord node as a rendezvous point for a given key without any central coordination Project (2010). Chord algorithm does not particularly specify any means for storage of the resource; this is done by DHash which is built on top of Chord and also handles storage of data blocks on the active nodes reliably Project (2010). This is achieved using techniques like replication and erasure coding. The logical application interface for DHT based systems is defined as: $\text{Key} = \text{put}(\text{data})$ and $\text{Data} = \text{get}(\text{key})$ Project (2010).

Pastry

Pastry is completely decentralized, scalable and self organizing network which dynamically adapts to the addition or removal of nodes Guvnec & Urdaneta (2010). Each node in Pastry Network has unique and random identifier called NodeId in a circular 128-bit identifier space. With a message and a numeric 128-bit key, a node can route the message to a node with NodeId which is numerically close to the key within the live Pastry Network Rowstron & Druschel (2001). This results in first order balancing of the storage requirements and query among the nodes in the Pastry network and also does not require global

co-ordination Rowstron & Druschel (2001).

Routing in Pastry for a given message it checks the following conditions Guvnec & Urdaneta (2010):

- If it falls within the NodeId's leafset then the message is directly forwarded to it.
- Else, the message is forwarded to a node that shares the most common prefix with the key using the routing table
- Else if the routing table is empty or the node is unreachable, then message is forwarded to node that is numerically close to the key.

If given N as number of live nodes in the overlay Pastry Network then expected number of forwarding steps $\mathcal{O}(\log N)$ and size of routing table for each node $\mathcal{O}(\log N)$ Rowstron & Druschel (2001).

CAN

CAN is also a distributed system which is DHT based that maps keys to values on big scale network like internet. As discussed above CANs basic idea is to build a hash table and the basic operations performed are insertion, lookup and deletion of the key, value pairs. In the CAN network each node stores a chunk (also called zone) of the total hash table. Moreover it stores smaller amount of information of adjacent zones Ratnasamy et al. (2001).

In CAN the network is formed in a tree like structure where each node is associated to one, at the parent level and to a group at a child level. When a query is made, it travels from the top most level going down through the network until the resource is discovered or until the last leaf is reached Guvnec & Urdaneta (2010). The architecture of the CAN is a virtual multi dimensional can be viewed as Cartesian coordinate space. CAN design centres around a virtual d-dimensional Cartesian coordinate space on a d-torus which is independent of the physical location and physical connectivity of the nodes Ratnasamy et al. (2001). The overall Cartesian coordinate space is dynamically partitioned among all the nodes such that each node belongs to one distinct zone with in the entire space Ratnasamy et al. (2001). To route a query, node maintains a routing table which holds the IP locations as well as the virtual coordinate zone of each of its neighbour. Using the co ordinates the message is routed towards destination.

CAN construction take place in three steps:-

1. A joining node must find a node which is already on the CAN network
2. Using the CAN routing mechanism, it must find a node whose zone will be split
3. Lastly, the neighbours of split zone are informed.

Tapestry

Tapestry is another P2P structured overlay network which provides high performance, scalable as well as location independent routing of the messages. It uses adaptive algorithm with soft state to maintain fault tolerance with regards to changing node membership and network faults. Tapestry provides decentralized object location and routing (DOLR), the DOLR interface provides routing of messages to end points like nodes or object replicas Zhao et al. (2004). Each Tapestry node is assigned a unique id and more than one node can be hosted by a single physical host. Tapestry utilizes identifier space of 160 bit values with a 40 digit key. The efficiency of the Tapestry increases with the increase in the network size. Moreover to allow multiple applications every message contains an application specific identifier which helps the node to select a process or delivery of message to a specific port Zhao et al. (2004).

Table 2.1 shows the classification of P2P routing infrastructures in terms of their network structure, with typical examples. Table 2.2 summarises infrastructure for routing and resource discovery location.

	Centralisation		
	Hybrid	Partial	None
Unstructured	Napster	Kazaa, Edutella	Gnutella
Structured			Chord, CAN, Tapestry, Pastry

TABLE 2.1: A classification of P2P routing infrastructures in terms of network structures
Source: Androutsellis-Theotokis & Spinellis (2004)

P2P Infrastructure	Description for Routing and Location
Flooding	Infrastructure that provides functionality for searching “blindly” on overlay networks.
Chord	A scalable peer-to-peer lookup service. Given a key it maps the key to a node.
CAN	Scalable content addressable network. A distributed infrastructure that provides hash-table functionality for mapping file names to their locations.
Pastry	Infrastructure for fault-tolerant wide-area location and routing.
Tapestry	Infrastructure for fault-tolerant wide area location and routing.

TABLE 2.2: Summary of infrastructure for routing and resource discovery location Source: Androutsellis-Theotokis & Spinellis (2004)

2.2 Resource Discovery and Routing

Table 2.3 compares various features of routing algorithms used in P2P systems.

2.2.1 Resource Discovery in Unstructured P2P Systems

In unstructured P2P systems for instance Gnutella, various nodes(peers) are organised into a random graph where the edges of the graph are the links between various nodes this constructing an overlay network Chawathe et al. (2003); Forum (2002). Flooding technique is used for routing a query through the overlay network. Upon query, the visited node compares the query against its shared resources and is then requested to forward the query to its neighbours. This system of resource discovery is highly robust and offers vast improvement on factor of scalability as compared to Napster or other centralised search systems but suffers from an expensive cost of saturation of overlay network due to large bandwidth consumption Chawathe et al. (2003); Forum (2002); Aberer et al. (2004); Napster (2003). Various techniques have been introduced to improve the efficiency of this system that includes random walks, informed searches, and node grouping Bawa et al. (2003); Zhu & Hu

Feature	Conventional Flooding	Random Walks	DHT	Range Query
Infrastructure	Performed on unstructured P2P networks	Performed on unstructured P2P networks	Performed on structured (DHT) P2P networks	Performed on structured (DHT) P2P networks
Scope	Works same with any network	Works best with multiple queries and peer clustering	Works same with all the DHTs	Works best when semantic proximity of keys is maintained
Search Complexity	On average search is done in $k*N$ time (k = average degree of nodes, N = total number of nodes)	On average search is done in log time	On average search is done in log time	On average search is done in log time
Relevance and Results	Returns single result	Returns single result	Returns single result	Returns a set of results
Cost Associated with Resource Discovery	Very wasteful of resources, as every peer processes each query	Less taxing on resources	Not too taxing on resources	Min-max algorithm uses resources wisely
Response Time (Routing and Searching)	Is not very fast, as every peer processes each query	Result are reasonably fast	Routing is very fast	Shower algorithm is very fast

TABLE 2.3: Comparison of features of routing algorithms Source: Prakash (2006)

(2006); Lv et al. (2002); Crespo & Garcia-Molina (2002).

Random walks were introduced to improve the issue of saturation by introduction of techniques *time-to-live* (TTL) and *checking* Lv et al. (2002). Like flooding, random walks, is uninformed search technique where the query is randomly forwarded to nodes. As an answer to saturation of the overlay network, the total number of nodes to be visited is defined using TTL. Also, *checking* technique is used where before forwarding to next node, the query originator is “checked with”. These techniques of controlled flooding refined resource searching mechanism but suffered from lack of results due to restrictions imposed by TTL.

To increase the effectiveness of search mechanism, informed searches were introduced that offered improvement in performance by using information on nodes and their resources Lopes & Botelho (2008). This information is collected as part of previous queries. Crespo et. al. introduced the technique *routing indices* (RI) for informed searches, where queries are routed to nodes that were more likely to provide a resource Crespo & Garcia-Molina (2002). In this technique uses distributed-index mechanism that maintains indices on each node. Given a query, the RI data structure returns a list of ranked nodes for forwarding a query. In informed searches, propagating a query to nodes where there is likeliness of discovering a resource help reduce the network load because of less flooding.

Other resource location techniques such as *SETS* and *ESS*, are based on a concept of grouping content to organise nodes Bawa et al. (2003); Zhu & Hu (2004). The search in SETS is based on topic-segmentation of overlay network. In other words, SETS partitions nodes into topic segments such as nodes with similar content belong to same segment Zhu & Hu (2006). SETS suffer from single point failure and hence has performance bottleneck Zhu & Hu (2006). ESS is based on information retrieval algorithms to perform resource discovery on Gnutella-like P2P systems. As in SETS, nodes with similar content are segmented into same semantic group Zhu & Hu (2004). The concept used by ESS is to place indexes of semantically close files into same nodes with high probability of exploiting information retrieval algorithms and locality sensitive hashing Zhu & Hu (2007).

A multiple keyword based searching technique called *local indexing* is used for locating resource using multiple keywords Tang & Dwarkadas (2004). As seen in Figure 2.3(i), the record of terms contained in each resource is stored on that particular node. Upon query, the search keywords are forwarded to each node using flooding technique, where they are compared for relevance. This technique is effective for getting better search

results but suffers from classical saturation factor on overlay network.

2.2.2 Resource Discovery in Structured P2P Systems

Structured P2P systems have been proposed to provide a more scalable solution as compared to first generation unscalable unstructured P2P systems. In structured systems, a node is associated with keys and their values. When a query is presented it is changed into the search for the key. The hash table on the peer is used pass the query forward to other peer whose address is numerically closer to requested key. The examples of structured systems are Chord, and CAN Ratnasamy et al. (2001); Stoica et al. (2001). In hybrid systems for instance Pastry, the routing structure is comparatively more fluid as compared to Chord as the routing table can suggest the routing of the query to any node that is part of the defined subspace Talai et al. (2006); Rowstron & Druschel (2001).

Structured systems perform better than unstructured systems with respect to scalability, as DHT has many advantages, such as scalability, load balancing, logarithmic hop routing, fault tolerance, and self organising nature Singh et al. (2009). Although self-organising works as the advantage but as each peer must periodically update all its neighbours and hence results in increased traffic Mastroianni et al. (2005). When the nodes leave or join the network the updated index need to be redistributed and hence the tables need to be restructured. This is not the case in unstructured systems as node can leave or join the network without sending stabilisation message. Unstructured systems have provided many strategies for reducing traffic like dynamic querying, routing indices, and super-peers architectures Chawathe et al. (2003); Karnstedt et al. (2004). Structured systems have advantage over unstructured systems as these systems provide ability to route the queries in very small number of hops. DHT-based systems are known for exact-match lookups, given a query both Chord and Pastry resolve the queries in $\mathcal{O}(\log(n))$, while CAN requires $\mathcal{O}(n^{\frac{1}{d}})$ steps, where n is number of nodes and d is number of dimensions in CAN Stoica et al. (2001); Ratnasamy et al. (2001). As the peers and the resources are based on the hash function – key generated by the hash function is very specific Stoica et al. (2001); Ratnasamy et al. (2001). As the queries may not be exact, it may be difficult to find the resource in the structured network Mastroianni et al. (2005); Singh et al. (2009).

However, in *keyword-search* the queries do not have to be exact and can comprise of multiple-keywords. The information retrieved in such scenario consists of a set of resources

that match the criteria given as a query. The proposed system that support keyword-search on top of DHT-based structured P2P system are categorised by their indexing technique viz. *global indexing* [Li et al. (2003); Reynolds & Vahdat (2003); Casey & Zhou (2009); Tang & Dwarkadas (2004)], and *hybrid indexing/optimised-hybrid indexing* [Zaharia & Keshav (2008); Tang & Dwarkadas (2004); Chen et al. (2008)].

In *global indexing* as seen in Figure 2.3(ii), the inverted list record is maintained on every node - information about nodes that contain a particular term. Upon query that contains multiple keywords, the query is routed to node containing that keyword. Then the inverted lists are intersected to find resource that contains the requested keywords. This largely reduces the number of nodes that need to be visited, however large amount of communication is introduced during intersecting phase. Moreover, communication cost grows with increase in length of inverted list Tang & Dwarkadas (2004); Zhu & Hu (2007).

In *hybrid indexing* as seen in Figure 2.3(iii), each node holds the complete inverted list of terms describing the resources on that node and also the inverted list of terms that are forwarding terms for resources shared on this node. Given a multiple keyword based query, the query is routed to node containing the search keywords. Then, this node performs a local search without connecting to other nodes about list of forwarding nodes by querying the inverted list of each found resource on this node. The efficiency of this type of indexing is higher than that of global indexing but suffers from increased cost of publishing term data Zhu & Hu (2006).

In *optimised hybrid indexing* (See Figure 2.3(iv)), the terms that describe a resource is published under resource's top terms (terms that are central to a resource) Tang & Dwarkadas (2004). Clearly, the search may be degraded because of limiting the publishing of keywords under resource's top terms Zhu & Hu (2006).

Another effective way for resource discovery process is to establish *semantic links* between the nodes that are based on node properties which are described by the resources shared by those nodes Sun et al. (2006); Kang et al. (2007); Crespo & Garcia-Molina (2004); Tang et al. (2003); Arabshian et al. (2009). In Kang et al., the semantics information is used for searching resources in a scalable manner Kang et al. (2007). A. Crespo suggests the semantic overlay network (SON) where the peers are organised based on logical similarity between the content Crespo & Garcia-Molina (2004). Semantic information can be used to create P2P networks that are more organised than unstructured overlay and are capable of handling multiple keys for finding resource on network unlike structured overlays. Locality

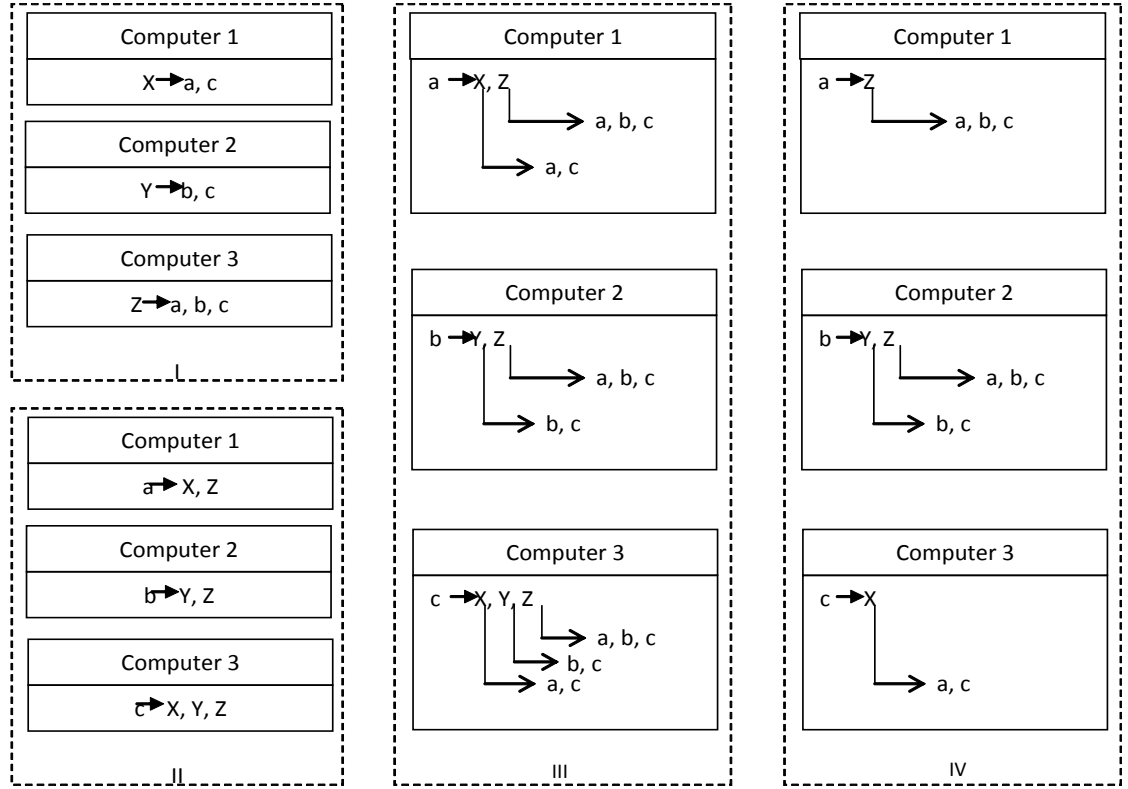


FIGURE 2.3: Comparison of distributed indexing structures. (i) Gnutella-like local indexing. (ii) Global indexing. (iii) Hybrid indexing. (iv) Optimized hybrid indexing. a , b , and c are terms. X , Y , and Z are documents. Source: Tang & Dwarkadas (2004)

awareness is another version where the peers are organised based on matching tags that are used to describe a resource Sun et al. (2006). pSearch introduces the concept of semantic overlay on top of a DHT based structured P2P system Tang et al. (2003). In this overlay, the resources are organised based on their semantic vectors (such as distance). pSearch proposed to integrate semantic storage and retrieval capabilities into CAN, where resource index is stored by using its vector representation as coordinates Zhu et al. (2003). GloServ uses a keyword-based search on a hierarchical hybrid P2P network to build semantic overlay between nodes that operate in the same domain Lopes & Botelho (2008); Arabshian et al. (2009). Even though this attempt at creating semantic links between nodes and resources may help improve the resource discovery, but no test results have been published yet by the authors.

Both structured and unstructured systems heavily rely on stationary software modules. These modules keep track of all resource discoveries. They use the host computer resources and can potentially drain the local resources and may cause failure of host computer. Backbone of both approaches is P2P communication. P2P communication blurs the distinction between client and server computers. This can potentially saturate the network. Unstructured resource discovery has a linear connection between computers where each computer knows the ping computer. Failure of any computer in the chain results to loss of all down stream resources.

2.2.3 Resource Discovery in Mobile Agent Systems

As an alternative to stationary software modules, multi-agent systems offer following merits that make mobile agents in particular suitable for resource discovery in P2P systems Dunne (2001):

- Asynchronous: After a mobile agent is dispatched, there is no need for the creator peer to keep track of mobile agent. The thread can be completely released. Theoretically speaking, the creator peer does not even need to remain connected to a network. A mobile agent will perform the given tasks completely in parallel with the creator peer as a separate thread. After all of the tasks have been fulfilled, mobile agent will return to the creator peer (when it is connected to the network).
- Autonomous: Mobile agents can compute its itinerary as it progresses through the network. It is able to choose the next site according to conditions it has learnt about,

and history of visited peers and current peer. Mobile agents may also visit peers that were unknown when it was originally dispatched, which in particular suits network based resource discovery.

- **Compatibility:** Agent based systems can be combined with successful features from other resource discovery systems.
- **Bandwidth Consumption:** The mobile agents for resource discovery require lesser bandwidth. As opposed to the multiple interactions between peers, mobile agent packs these interactions and sends them as discrete piece of traffic. Also mobile agents are much smaller in size and grow dynamically as they accommodate more data. In structured or unstructured systems, the communication is synchronous which is not the case with mobile agent which can encapsulate its state and carry on the execution on the different node asynchronously Bellifemine et al. (2007).

Dasgupta et al and Kambayashi et al introduced multi-agent systems (MAS) for resource discovery Dasgupta (2003); Kambayashi & Harada (2009). Both systems are inspired from ant communities for development of their P2P system. They use *Anthill* MAS that emulates the resource coordination behaviour as observed in ants Babaoglu et al. (2002); Babaoglu & Jelasity (2008); Yang et al. (2007). In this MAS P2P system resources are known as nests and user request to locate resources is carried out by ants. Upon query, the ants visit various nests on overlay network. Ants restrict from communicating to each other but leave information about the service they are implementing in the resource manager found at each nest site. The behaviour has analogy to pheromones that has advantage of allowing network to self-organise over a period of time Lopes & Botelho (2008). Ants greatly improve upon the flooding issue raised in unstructured P2P systems as only one ant visits the nest at one time. The next nest chosen for ant to visit is either deterministic or random, which means that search performance may be slow. This is observed in [Dasgupta (2003) and Kambayashi & Harada (2009)], where overlay network becomes more “knowledgeable” over a period of time. To improve upon this disadvantage, Kambayashi et al build their P2P system on top of structured P2P system called Chord Stoica et al. (2001); Kambayashi & Harada (2009). Mobile agents (ants) in their system may use <key, value> map to find resource in cases when deterministic path cannot be calculated. Kambayashi et al also use indexing (TF-IDF) to calculate logical distance between two nests based on correlation between keywords shared between nodes. The correlation is calculated using primitive form of Jaccard similarity.

2.3 Critical Review

For the research work, the author understands from the literature survey that semantic links between the nodes is useful for resource location and for node coordination - to be used for deterministic routing of the query which is also one of objectives of this research work. The author further understands that MAS and mobile agents offer nodes a greater degree of autonomy as they can migrate to new nodes based on information provided by visited nodes and hence offer relevant results to user. It is further understood that search load can be reduced by reducing number of messages or number of hops made by mobile agent during migrations from one node to another. The author aims to exploit heterogeneity of resources hosted by nodes on overlay network to locate resources in minimum number of hops i.e. drive/route the mobile agent on overlay network based on the content hosted by nodes.

2.4 Agent Based System Development Frameworks

This section provides review of the different mobile agent platforms and justifies the choice of the mobile agent platform - JADE Schoeman & Cloete (2003); Trillo et al. (2007):

Mobile Agent System Interoperability Facility (MASIF) standard OMG organization defined a standard named as Mobile Agent Framework (MAF) (later on changed to MASIF), which is aimed at promoting the interoperability of JAVA based mobile agent systems developed by different vendors Zhong & Liu (2003). MASIF presents a set of definitions and interoperable interfaces for mobile agent systems. The *MAFAgentSystem* interface and the *MAFFinder* interface are the two primary ones which are designed towards the following interoperability concerns Schoeman & Cloete (2003):

1. Management of agent, including creation, suspension, resumption and termination;
2. Commonly accepted mobility infrastructure that enabling the communications between different mobile agent systems and the transport of mobile agents;
3. A standardised syntax and semantics for naming services; and
4. A standardised location syntax for finding agents.

MASIF also excludes the following important architectural components in its standardisation attempts Schoeman & Cloete (2003):

- (1) It only addresses interoperability between agent systems written in Java, thus brings the obstacle of the interoperability between non-Java based systems and MASIF compliant systems;
- (2) It does not address local agent operations such as agent interpretation and execution;
- (3) Some conventional issues of inter-agent communication are excluded Milojevic et al. (1998).

Foundation for Intelligent Physical Agent (FIPA) is the standards organisation for agents and multi-agent systems who promotes agent-based technology and the interoperability of its standards with other technologies Vieira (2001). A collection of specifications have been provided, which are intended to promote the interoperation of heterogeneous agents and the services they represents. However these specifications are focussed on agent communication languages, agent management, message transport and the support for the use of ontologies in general.

2.5 Qualitative Comparison of Mobile Agent Platforms

The following are the most popular mobile agent platforms Schoeman & Cloete (2003):

JADE Specification of FIPA are implemented by Java Agent Development Framework (JADE) that provides Application Programming Interfaces (API) for Java based implementation of multi-agent systems Bellifemine et al. (2007). The agent platform can be distributed on multiple hosts. Each platform only hosts one application and hence only one Java Virtual Machine (JVM). JVM can allow several agents to execute concurrently on the same host. The *Agent* interface is the primary interface that concerns is implemented for all types of agents. JADE implements the complete Agent Management specifications suggested by FIPA including services such as Agent Management System (AMS), Directory Facilitator (DF), Message Transport Service (MTS), and Agent Communication Channel (ACC). In addition JADE has implemented Agent Communication stack, ranging from FIPA-ACL for message structure and FIPA-SL for message content and other FIPA interaction and transport protocols Bellifemine et al. (2007).

The main drawback is that currently inter-platform mobility service is being developed and not available to researchers. Also, there are no proxies and agent searches the current location of its target by querying the AMS.

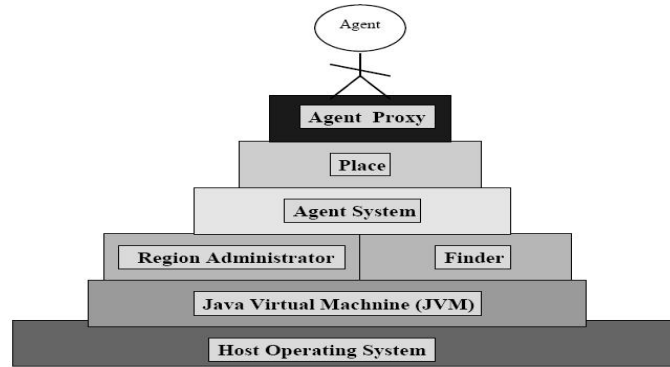


FIGURE 2.4: The SMART architecture Source: Wong et al. (2001)

Voyager is a commercial mobile agent platform supporting dynamic aggregation feature. The basic idea behind Voyager and dynamic aggregation feature is to reuse existing Java classes and make objects of such classes mobile by means of incorporating those objects as its attachments (known as facets) and move from one site to another hence moving those objects with itself. The objects will retain their internal state upon moving from one host to another Wong et al. (2001). The main focus is on the management of remote communications of traditional Common Object Request Broker Architecture (CORBA) and RMI protocols. It also offers dynamic generation of CORBA proxies and mobile agents. Agents communicate via RMI using proxies.

The main drawback of Voyager is that it is commercial product and is not freely available.

Scalable Mobile and Reliable Technology (SMART) SMART Wong et al. (2001) is a MASIF specification compliant client-server based mobile agent platform. As Figure 2.4 shows, there are four main components in smart architecture Wong et al. (2001): Region administrator, which uses a finder model to provide naming services to the region administrator and also to the agent system; Agent system, enables mobile agents to create, migrate and destroy themselves; Place, forms the execution environment; and Agent proxy, provides the mobile agent API for applications written in SMART.

The main disadvantage of SMART is that it does not support agent communication as described in MASIF standard. Also, it does not provide good security mechanisms.

D'Agents (Robert S. Gray) is a general purpose mobile agent system which was developed to support distributed information retrieval and to support for strong mobility and

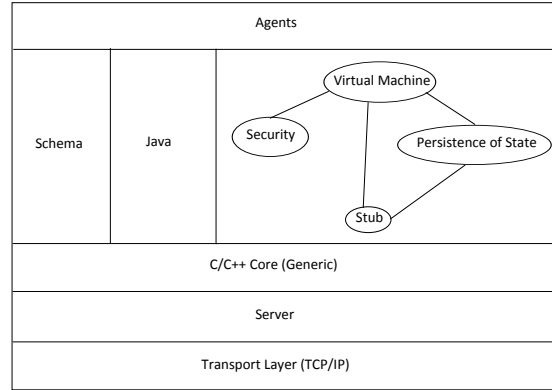


FIGURE 2.5: The D'Agent architecture

multi-agent languages. Using D'Agent, several information-retrieval applications, ranging from searching three-dimensional drawings of mechanical parts for a needed part to supporting the operational needs of a platoon of soldiers have been implemented. The architecture of D'Agent is shown in Figure 2.5. TCP/IP is used to provide transport mechanism. Server layer is a multi-threaded process and runs multiple mobile agents as threads inside a single process. The Generic C/C++ core layer holds shared C++ libraries used by agent threads. The upper layer provides the execution environment for Java, Tcl, or Scheme. The agents themselves are defined on the top layer Schoeman & Cloete (2003).

The disadvantage is that for deployment using Java platform the virtual machine (VM) needs to be extended instead of agent server that resides on top of VM.

Grasshopper is an OMG MASIF and FIPA-conformant agent platform, which consists of a Distributed Agent Environment (DAE) and a Distributed Processing Environment, as Figure 2.6 shows. A host in the distributed agent environment include an agency that has access to the services including execution, transport, management, communication, security, naming mechanism, adapter interfaces for external hardware/software, task control functions, and application-specific GUIs Schoeman & Cloete (2003). The distributed processing environment is composed of following components: Regions, facilitates the management of the distributed components (agencies, places, and agents) in the Grasshopper environment; Places, provides a logical grouping of functionality inside an agency; Agencies, as well as their places can be associated with a specific region by registering them within the accompanying region registry; and Different types of agents – mobile agents and stationary agents. Mobile agents move from one platform to another, whereas stationary agents reside

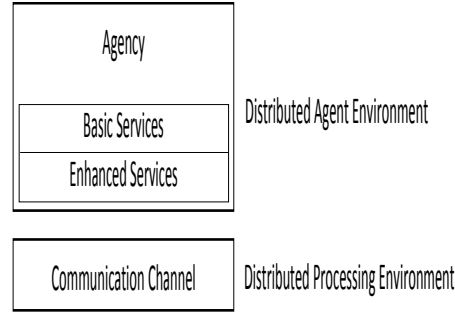


FIGURE 2.6: The simplified version of Grasshopper Architecture. The basic services include MASIF and Core Services. MASIF includes agent creation, destruction, suspend, activate and location services and Core services include agent execution, transport, management, communication, security and naming. The enhanced services include APIs, GUIs and task control features.

on one platform permanently (Grasshopper Mobile Agent Platform).

The main disadvantage of Grasshopper is that it is not available anymore and new versions will not appear in the future. The region server could become a bottleneck, as it must update every proxy right before using it Trillo et al. (2007).

Aglets (Aglet) is a well known Java based mobile agent platform, which contains libraries for developing mobile agent based applications. This platform follows MASIF specification Trillo et al. (2007). Aglets are built around single-thread model for agents and a communication infrastructure based on message passing. Both synchronous and asynchronous messages are supported Trillo et al. (2007). Agents in Aglets use proxies as abstraction to refer to remote agents for sending messages that is similar to stubs in Remote Method Invocation (RMI). As Figure 2.7 shows, Aglets' architecture consists of two layers: Runtime layer, consists of a core framework and sub-components to provide services such as serialization/de-serialization, class loading and transfer, reference management and garbage collection, persistence management, maintenance of byte code, and protecting hosts and agents from malicious entities; and Communication layer, defines the methods for creating and transferring agents, and tracking and managing agents in an agent-system-and-protocol-independent way Schoeman & Cloete (2003).

The drawback of this platform is that the proxies it provides are not dynamic proxies and hence cannot be used when the agent has migrated which means in case of using it again, the proxy has to be updated manually. Single thread model is also an issue as in case of synchronous messages being sent by one agent to other agent at the same time can result

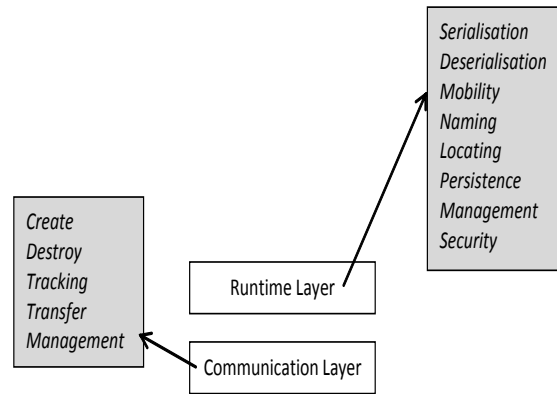


FIGURE 2.7: The Aglets architecture Source: Schoeman & Cloete (2003)

in a deadlock.

Table 2.4 summarises the main features of mobile agent platforms.

Table 2.4: Qualitative Comparison Among Mobile Agent Platforms Source: Trillo et al. (2007)

Model	Behaviours	Events	Procedural	Procedural	Procedural
Elements	Containers Main containers Platform Agents DF, AMS, MTS	Contexts Agents (aglets) Tahiti	Servers Agents	Regions Agents Places	Regions Agents Places
Proxies	No	Yes	Yes	Yes	Yes
Dynamic proxies	No	No	Yes (for- warding)	Yes	Yes
Synchronous communica- tion	No	Yes (deadlocks)	Yes	Yes	Yes

Feature	JADE	Aglets	Voyager	Grasshopper	SMART
Available to download	Open Source	IBM Public Licence	Not Free	Open Source	Open Source
Asynchronous communication	Yes	Yes	Yes	Yes	Yes
Messages	Yes (FIPA Standard)	Yes	No	Yes (FIPA)	Yes
Remote calls	No	No	Yes	Yes	Yes
Callbacks after migration	No	No	Yes	No	No
Call/messages by name	Yes (Agent Identifier)	No	No	No	No
Migration by name	Yes (AMS)	No	No	No	No
GUI tools	Yes	Limited	No	No	No
Level of activity	Very High	Very Low	Medium	None	None
Security mechanism	Yes	Basic	Yes (security managers)	Basic	No
Some other features	Ontology Support, FIPA Compliant	Itinerary Setup	Multicast Publish and Subscribe	MASIF FIPA	MASIF

JADE was one of the first FIPA-compliant platforms developed. JADE offers an agent runtime system and a predefined programmable agent model and of a set of management and testing tools that are missing features in other platforms. It simplifies the development of applications that requires negotiation and coordination that is one of the highlights of MAS system developed as outcome of this project. With use of ACL and mail-

boxes for each agent, developers steer clear of remote method invocations where the remote references require updating upon migration - a facility that is required for mobile agent. Not that it is required within the scope of this research work but due to JADE's compliance with FIPA specification end-to-end interoperability between agents of different platforms is possible. JADE's API is independent from underlying network and Java version and is standard across Java Enterprise Edition (J2EE) and Java Mobile Edition (J2ME) that allows reusability of application code. Also, JADE has ontology support where this work can be extended for future work.

2.6 Summary of the Chapter

In this chapter, the author has collated and researched the information in the field of resource discovery on unstructured, structured, and MAS systems. Also, the author has categorised the resource discovery techniques used in various types of overlay networks. This researched information provided insights into resource discovery systems and clearly characterised the properties that such systems should be attributed with. Based on these insights, in Section 2.3, the author proposed the characteristics that a successful resource discovery system should have for achieving maximum search result relevance with minimum search load and messages on the overlay network. In the penultimate section, the author discussed various mobile agent platforms - their features and drawbacks and also justification for use of JADE platform for development of MAS system.

In next chapter, design features and implementation of the proposed system are provided. The details about implementation of proposed design features are presented in program listings section.

CHAPTER 3

DESIGN FEATURES AND IMPLEMENTATION

The chapter details about system architecture and design features that implement the proposed characteristics of a resource discovery system using mobile agent. The author has conceived multi-agent resource discovery system using mobile agent called Affinity that

1. Captures the features of clustering of peers based on semantics of content shared,
2. Handles multiple keys to locate a resource by use of LSI similarity, and
3. Finally reduce the bandwidth consumption by providing mobile agent with ability to negotiate with peers regarding finding next site for migration and matching resource hosted by peer to user query under given constraints from user.

The features have been divided into sections and each section of this chapter discusses that feature and its realisation. The final section provides snippet of mobile agent communication - agent communication language, based on FIPA standards implemented using JADE and also implementation details of proposed features using JADE and Java. Detailed information about code based implementation and its deployment are found in program listings section of the thesis.

3.1 The Proposed Multi-Agent System for Resource Discovery - Affinity

3.1.1 The Proposed Global System Architecture

The architecture for the conceived system is illustrated in Figure 3.1. As articulated in the figure, the system has four layers - interface layer, reconnaissance layer, directory and resource layer and visiting agents layer. Each layer contains agents dedicated to perform certain task (detail specifications of agents are provided in Section 3.1.2).

The purpose of each layer is as follows:

- *Interface Layer*: This layer contains the interface agent that is used by the client to interrogate the system. The goal is to capture the requirements or needs of the user and respond back to them appropriately. User's interaction with system is through interface agent that helps in realisation of the given task. The request from user i.e. the search query facilitates the function of reconnaissance layer. The additional function of transformation of the submitted user request into a feature vector is also realised in layer.
- *Reconnaissance Layer*: This layer contains the reconnaissance agent that is created as a result of submitted query in the interface layer. The function of this layer is to temporarily contain the new created mobile agent while it communicates to stationary agents in directory and resource layer for node address where it can migrate to in order to realise the submitted query.
- *Directory and Resource Layer*: The function of this layer is to receive requests from reconnaissance agent, process them and return the results. This layer holds two stationary agent - local agent and information agent and is responsible for managing the data associated to shared resources on the node and multiple sources of node addresses that are semantically similar to content shared on this node. The task of determining appropriate node address and hence deterministic route to the node that hosts resource similar to given query is completed in this layer. The management of directory of shared resources on this node that are transformed into feature matrix after indexing is the function of local agent and the management of list of peers that are semantically similar to content of this node is done by information agent. The functionality to achieve autonomy is also achieved on this layer where information agent communicates to bootstrap server about its status every 300,000ms.
- *Visiting Agent Layer*: The function of this layer is to provide platform for the migrated reconnaissance agent that is visiting a particular node. This layer is a class that is capable to provide functionality of sending messages to and receiving messages from directory and resource layer of the visited node. This layer also provides additional functionality of query matching by collaborating with directory and resource layer for

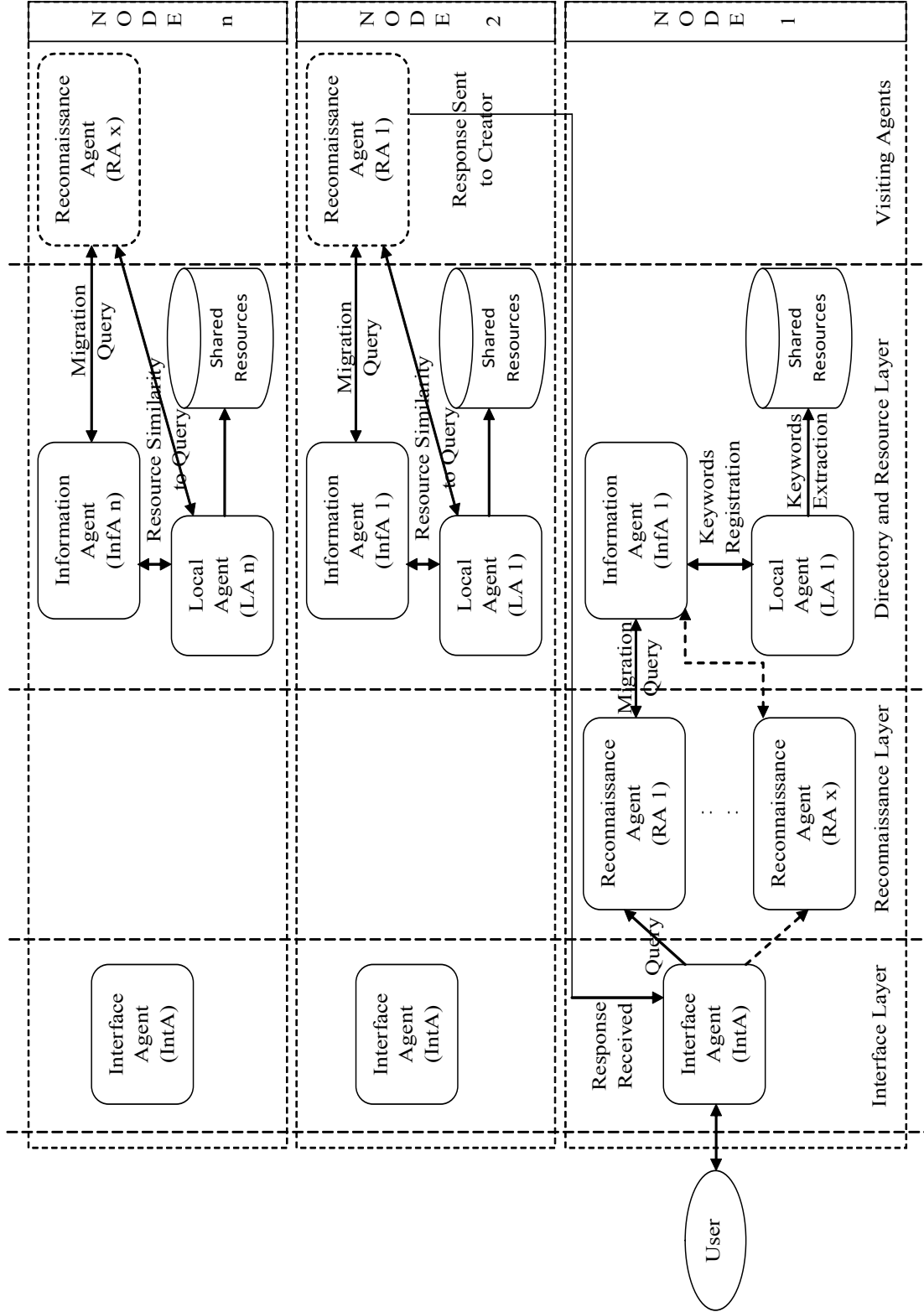


FIGURE 3.1: Global architecture of the system

realising the task of finding resource(s) hosted on this node that is semantically similar to submitted query.

3.1.2 Specification of Agents

The proposed system - Affinity is hybrid system based on the semantic overlay network, unstructured P2P system, and MAS. All peers share their resources that are maintained by the set of collaborating agents on each peer. The collaborating agents on each peer are

1. *Interface Agent (IntA)*,
2. *Local Agent (LA)*,
3. *Information Agent (InfA)*, and
4. *Reconnaissance Agent (RA)*.

The purpose of each is as follows:

- *Interface Agent*: IntA is a static agent that provides user interaction to the system. The user interacts with IntA using the GUI interface that a.) shows search query, b.) informs search results, and c.) inform active RA(s).
- *Local Agent*: LA is another static agent that holds information i.e. keys for defining local resources and the corresponding location of resource on the peer. In addition, it has tasks to serve InfA for keywords request and RA for keyword similarity. Local indexing of shared resources is maintained by LA.
- *Information Agent*: InfA holds information about peers that are semantically similar to this peer i.e. the indexing results propagated by bootstrap server are maintained by this agent. It holds a data structure that contains all peer's GUID, similarity value and keywords that it is sharing. InfA is responsible for computing routes for RA upon request of migration query. InfA also communicates to LA to request a list of keywords that a peer is sharing that it in turn is submitted to bootstrap server for registration and finding peers that belong to same cluster.
- *Reconnaissance Agent*: RA is a mobile agent for resource discovery; that is created by the IntA upon user's search request. RA migrates to new peers by requesting node

address from InfA. RA's task is to migrate to peers and to investigate LA that is responsible for hosting resources (hence keywords) about their possible similarity to user's query and report it to IntA.

In addition to the proposed multi-agent system, the overlay network organisation of this P2P system is improved by InfA registration to bootstrap server. A *bootstrap* server maintains a list of peers that are currently in the system. Upon *registration/joining*, the bootstrap server replies with list of peers that are semantically similar to this peer.

As detailed in Section 3.2.1, the cosine similarity between peers is actually keyword similarity of hosted resources of those peers. The result of this similarity is cluster effect as illustrated in Figure 3.2. Although, the sparsity of keyword matrix on the bootstrap server is large but still it is overlooked by potential advantage, that each peer is now organised in overlay network (i.e. it only knows the address of neighbours in a cluster). The Globally Unique Identifier (GUID) of neighbours in cluster are used to prepare a hash table that is maintained by InfA. When a neighbour *disconnects* from overlay network, it informs bootstrap and its neighbours to remove its GUID from matrix and hash tables respectively. Upon *creation* of RA, it communicates to InfA to provide it with itinerary (next site) for migration. InfA uses the hash table provided it by bootstrap server to issue a peer GUID that host resources/keywords that are close to requested user query.

3.2 The Proposed Mobile Agent Routing

Peer clustering is based on the conceptual content of resources shared by peers. The objective is to organise an overlay network in such a way that when given a query, small number of peers are selected based on "higher" chance of query hit. The benefit of this strategy is two-fold. First in context of peer clustering - the peers to which RA migrates to will have many matches, so that the query is answered faster, and second in context of RA routing - the peers with "lesser" chance of getting a query hit will be steered clear by the migrating RA, thus avoiding wasting resources on that query (and allowing other queries to be processed faster). Peer Clustering and Agent Routing are accomplished using LSI. The following Section 3.2.1 explains the state vector and singular vector decomposition (SVD) based semantics for peers and keywords hosted.

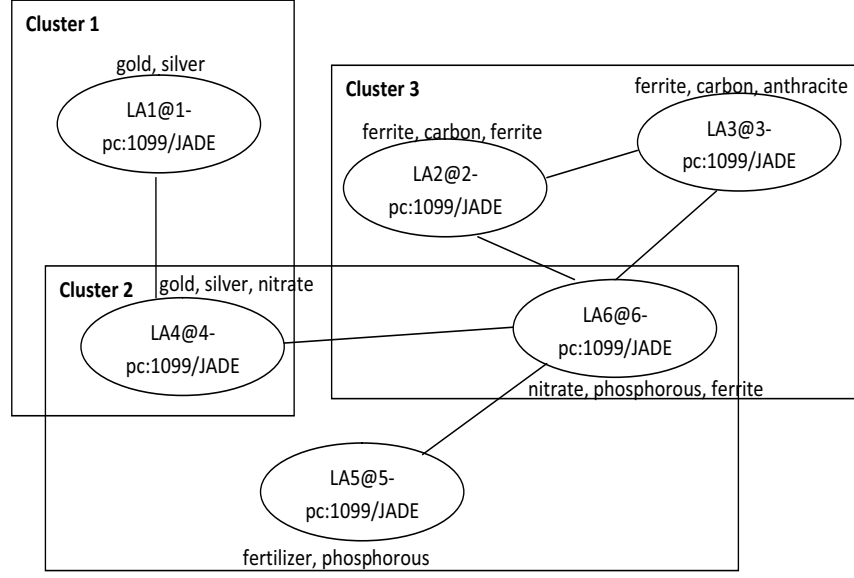


FIGURE 3.2: Peer clustering and overlay organisation achieved using *latent semantic indexing*

3.2.1 Latent Semantic Indexing and Singular Value Decomposition (LSI-SVD) for Peer Clustering and Mobile Agent Routing

Latent semantic indexing (LSI) is a variant of a vector space model, where low rank approximation to the vector representation of the corpus is computed Gao & Zhang (2005). LSI considers that latent structures may exist in documents that may not be visible and may very well be hidden due to variability in word choice Gao & Zhang (2005). Singular value decomposition (SVD) of the corpus is calculated to estimate the structure of lexicon usage across the documents.

The nodes may be represented by number of keywords (lexicons) that it shares. Hence, a set of nodes can be represented by a matrix called *keyword-peer* matrix A . The elements of the *keyword-peer* matrix represent the frequency of each keyword f on a particular node. Let N be the number of peers in a P2P network, and K be number of distinct keywords (lexicons). It should be noted that N can be resources when observed from RA-LA point of view, but generically the author assumes it as number of peers. The feature matrix called *keyword-peer matrix* is constructed as $A \in [a_{ij}]_{K \times N}$ where a_{ij} = frequency of the keyword i on node j . $A_{ij} = 0$ if the peer j does not contain the keyword i . Not all keywords appear on all peers and hence matrix A is generally it is a sparse matrix. Now, matrix

A denotes <peer, keyword> pairs in the network, which is the knowledge of correlations between peers and keywords. To properly characterise latent semantics and correlations between peers in LSI, that matrix A is factored into product of three matrices using SVD Golub & Loan (1996).

$$A = USV^T \quad (3.1)$$

$U^T U = I_K$ and $V^T V = I_N$, I_K and I_N are identity matrices of order K and N respectively. Matrix S is a diagonal matrix with elements $diag[\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_{\min\{K, N\}}]$, $\alpha_i > 0$ for $1 < i \leq d$, and $\alpha_j = 0$ for $j > d$, where d is the dimensionally reduced matrix. SVD is a low rank approximation of matrix A Golub & Loan (1996). SVD is used to find the singular vectors corresponding to k largest singular values which dominate the original matrix. Peers and keywords can be characterised by linear combination of singular values i.e. a k -dimensional point in the feature space spanned by k singular vectors Liu et al. (2004). Deerwester et al. (1990) shows the small dimensions are enough to express latent semantic i.e. $k \ll \min\{K, M\}$. The resulting singular vector and singular value matrices are used to map keyword-based vectors for peers and queries into a subspace in which semantic relationships from the *keyword-peer* matrix are preserved while keyword usage variations are suppressed Hasan & Matsumoto (1999). The reduced dimension decomposed matrix as a new pseudo-keyword-peer matrix is given by

$$A_k = U_k S_k V_k^T \quad (3.2)$$

where columns of U_k contains the eigenvectors of the $A_k A_k^T$ matrix or first k columns of matrix U and the rows of V_k^T are the eigenvectors of the $A_k^T A_k$ matrix or first k rows of matrix V^T . S_k is a diagonal matrix that has its diagonal elements with special kind of values of the original matrix Deerwester et al. (1990); Golub & Loan (1996). These are termed the singular values of A_k that has first k largest singular values.

In SVD representation of original vector space, $A_k^T A_k$ is a $N * N$ symmetric matrix for inner products between peer vectors, where each peer is represented by a vector of keyword weights. This matrix can be used for cluster analysis for collection of peers. Each column of matrix, $A_k^T A_k$ is a set of inner products between peer vectors in corresponding column of the matrix A , and every peer in the collection. The cosine similarity measure of peers i and j can be computed as follows:

$$sim(i, j) = \frac{\langle i, j \rangle}{|i||j|} \quad (3.3)$$

For information retrieval in K -dimensional space query Q is treated as another set of keywords and hence query Q becomes $q = Q^T U_K S_K^{-1}$ that is compared to the peer represented by $p = p^T U_K S_K^{-1}$. These equations present the coordinates of the vectors in the K -dimensional space and query-peer cosine similarity is given by

$$sim(q, p) = \frac{\langle q, p \rangle}{|q||p|} \quad (3.4)$$

All peers share the keywords that inform about the hosted resources. The similarity between the keywords shared by various peers forms a cluster of peers that are similar to each other and thus forming a cluster and in turn an organised overlay network. This also increases the efficiency of discovering a resource as number of hops that RA has to take to find a resource are decreased. In order to get list of peers, another parameter - *minimum support* is passed by user. The significance of this parameter is to give user a level of control over list of known peers by forming a “canopy” on known peers. The value of minimum support ranges between -1.0 to $+1.0$ where -1.0 explains ambiguity - list of all peers registered i.e. ignoring the similarity results, and $+1.0$ explains certainty - list of all peers that are exactly similar to this peer i.e. only peers that are sharing same keywords with same frequency.

RA's routing is directly affected by the minimum support value passed by user during acquisition of peer list i.e. lesser the value of minimum support, larger set of peer list and that means RA has larger number of ambiguous peers to choose from or vice versa. However, another value of *minimum support* for resource discovery and this time it means the similarity of query passed by user to the keywords shared by various peers in peer list allows RA to find the peer where it will migrate to.

The exact value of minimum support has not been established but through experimentation it is realised the initial value for peer registration can be $+0.1$ or higher and for resource location $+0.5$ and higher can provide suitable results.

The unstructured network is created at random where to locate/search for particular resources, the message has to be forwarded to number of times. If this is limited by N hops, where N is the number of nodes within the query message's reach, then query routing complexity on an unstructured P2P network is of the order of N , or $\mathcal{O}(N)$. On structured

networks, or the MAS that have underlying overlay network based on structured overlay the query routing complexity is typically $\mathcal{O}(\log(N))$, where N is the number of network nodes. This is because the size of routing table increases according to power of two hence each step cuts the distance to target resource by half thus resulting in a lookup complexity of $\mathcal{O}(\log(N))$ Doval & O'Mahony (2003). In the proposed case, suppose N is the number of nodes and m is the *minimum support* of a node that ranges as $0.0 \leq m \leq 1.0$, and N_D is the maximum number of nodes that are semantically close where $N_D \ll N$, then the complexity of query routing is given as $\mathcal{O}(N_D)$, when $m = 0.0$ and $\mathcal{O}(N_D^{-\log(m)})$, when $0.0 < m \leq 1.0$. As *minimum support* increases the number of nodes required to be visited by migrating RA decrease logarithmically, and when *minimum support* is 0.0, it means the RA has to visit all nodes N_D in this particular domain. It is seen that the query routing complexity for resource location is much more effective in our system as compared to structured and unstructured system because of informed migrations performed by the RA. The results are later justified in experimentation in Chapter 4.

3.3 The Proposed Multi-Agent Collaboration for Resource Discovery

The system starts by starting up a bootstrap server. The LA locates all the resources that are shared by the peer and preparing the keyword list that defines the resource. The InfA requests the LA to inform it about the keyword list that in turn is used by the InfA to register the peer on bootstrap server. This behaviour is a cyclic behaviour of InfA that is scheduled every 300,000ms. Upon registration, based on *minimum support* value, the InfA receives the peer list containing list of peers, their similarity value and keywords shared by those peers. User's request for resource location to the IntA is attributed by list of keywords that form a query, *minimum support* value for acceptable results, and number of hops that the RA can make. The detailed interactions between the collaborating agents are shown in Figure 3.3.

The resource discovery is carried out using following algorithm:

1. When query is passed by user to the IntA, the IntA in turn creates the RA for that specific query.
2. The RA is informed about query, minimum support, and number of hops by IntA.
3. The RA requests the InfA for peer name in order to create route for migration. The

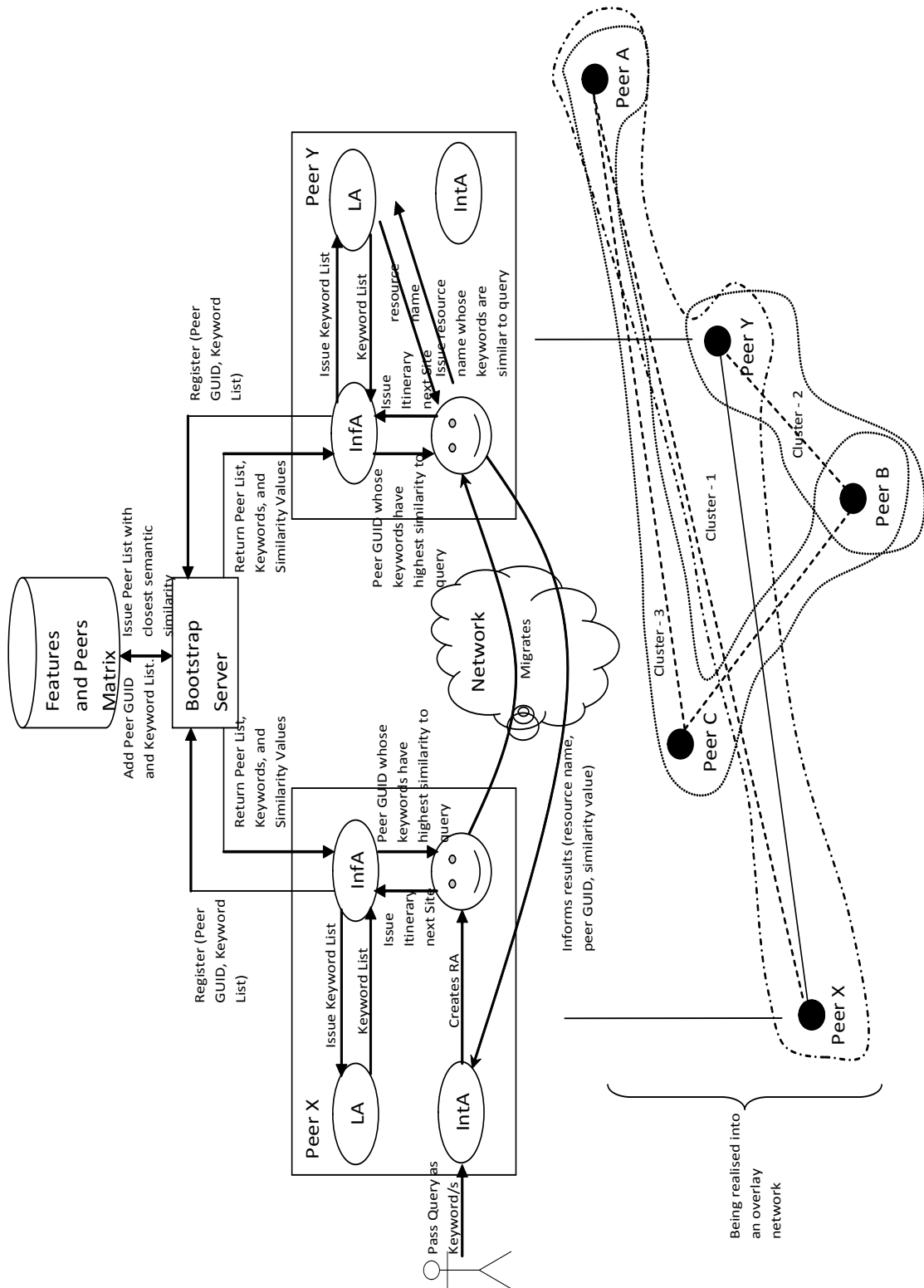


FIGURE 3.3: Interactions between multiple agents for resource discovery and realisation of an overlay network

peer name is informed by the InfA to the RA by looking up the peer name in hash table based on the semantic similarity of the query and the keywords shared by that peer.

4. The RA requests the agent management service (AMS) to find the container/platform where the selected peer is located.
5. The RA migrates to that peer and increments the number of hops by one.
6. The RA requests the LA of this peer to inform it about the resource name whose keywords are semantically similar to the query and higher than minimum support given by user. The LA provides the resource name to the RA.
7. The RA informs the IntA about located resource i.e. GUID of the peer where resource is located, resource name, cosine similarity value.
8. If number of hops made by the RA is less than maximum number of hops allowed by user then go to Step 9 otherwise, go to Step 10.
9. The RA requests the InfA of this peer for a new peer name where it should migrate to (hops to previously visited peers and to creator peer are not allowed). Go to Step 3.
10. As, number of hops made by the RA are equal to maximum number of hops allowed, the RA terminates itself.

Step 3 is shown in detail in Figure 3.4. The RA requests the InfA for GUID of the peer that it should migrate to; to find the resource. The InfA refers to the directory and calculates cosine similarity value based on degree of match between the query and list of keywords available. The highest similarity value is used to determine the peer GUID by looking up in the directory. Finally, the GUID of selected peer is informed to the RA. The RA now uses the GUID to find the container name from AMS where the corresponding peer GUID resides. The GUID of agents is generated based on container identifier and type of agent. This mechanism is better than “blind” or flooding technique as in this case the RA migrates with certain knowledge i.e. where and why to migrate to a certain peer as opposed to flooding the overlay network with communication messages or with multiple clones of the RA. Essentially, it improves the routing of the RA. The behaviour of the RA has been defined by *beforeMove* and *afterMove* methods. *afterMove* method is invoked just

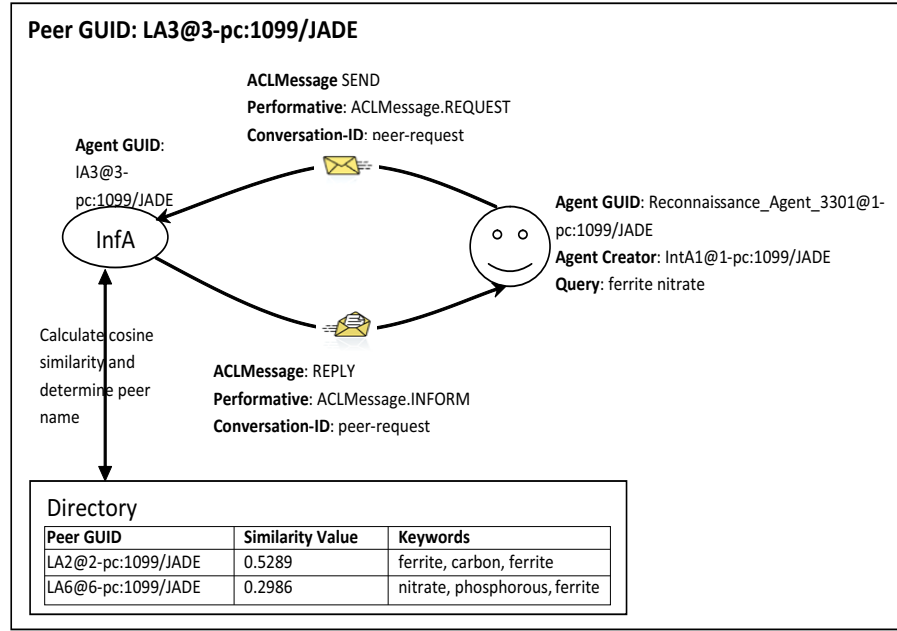


FIGURE 3.4: The RA's interaction with InfA for issuing new peer GUID

after migration to increment the number of hops made by the RA followed by checking the termination conditions.

Figure 3.5 presents the flow diagram for behaviour of the InfA and the LA when the RA arrives at a certain peer. Shown in the flow diagram are behaviours of three agents the RA, the LA, and the InfA. In addition to behaviours of agents, *blockedState* of the RA and *blockingReceive* of the RA is observed. These methods are invoked based on the ACLMessages in the mailbox of each agent. Essentially, as long as the RA has not received any message that matches the *MessageTemplate* (as seen in Figure 3.6), the RA is in blocked state.

3.4 Implementation

This section presents in detail various functions that have been implemented for realising the features viz. - feature matrix - indexing, clustering (nodes learning about other nodes), ranking and selection (nodes ranked and selected for routing of mobile agent), similarity, and behaviour of agents. The author has presented algorithm or pseudocode and its code based realisation details. Detailed implementation details can be found in Appendix C for

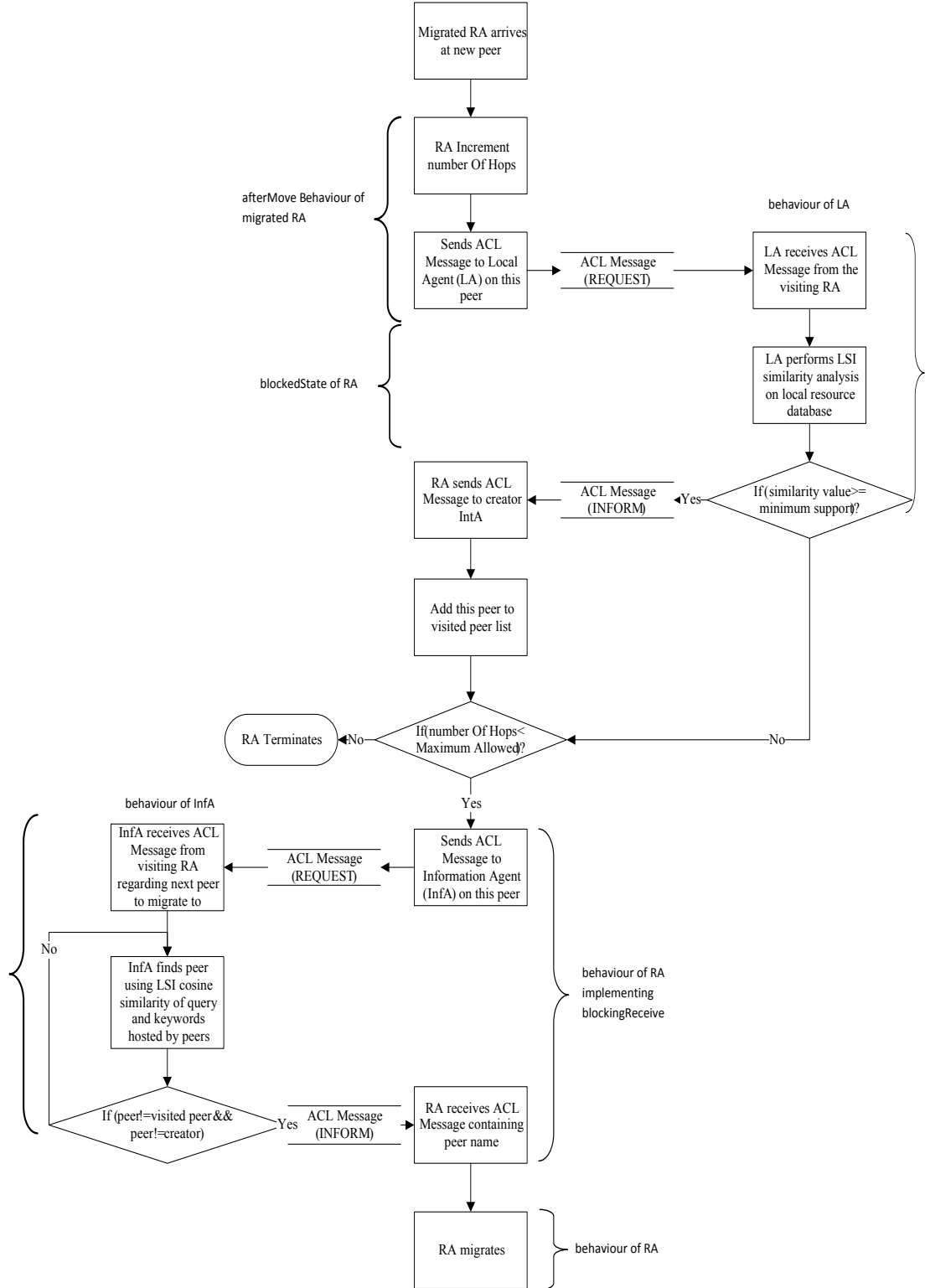


FIGURE 3.5: Flow diagram for behaviour of the InfA and the LA upon arrival of the RA

```

ACLMessage request=new ACLMessage(ACLMessage.REQUEST);
request.addReceiver(new AID(destLAName,AID.ISGUID));
request.setConversationId("search-request");
request.setReplyWith("request"+System.currentTimeMillis());
request.setContent(cont);
send(request);

private class ServeIncomingMessage extends Behaviour
{
    private MessageTemplate mt =
    MessageTemplate.and(MessageTemplate.MatchConversationId("search-
request"),MessageTemplate.MatchPerformative(ACLMessage.REQUEST));
    public void action()
    {
        try
        {
            ACLMessage request = receive(mt);
            ACLMessage reply = request.createReply();
            if(chosen!=null)
            {
                reply.setPerformative(ACLMessage.INFORM;
                reply.setContentObject(matchStore);
            }
            myAgent.send(reply);
        }else
        {
            System.out.println("No message yet");
            block();
        }
    }catch(Exception e)
    {
        e.printStackTrace();
    }
}
} //end class

```

FIGURE 3.6: *ACLMessage* from the RA to the LA for search request. *ACLMessage* received by the LA from the RA using *MessageTemplate* and replying with *setContentObject* or in blocked state.

Program Listing or media disc.

The author has used Java remote method invocation (RMI) and Java Agent Development Framework (JADE) (Bellifemine et al. (2007)) to implement the multi agent resource discovery using mobile agent system. Jade's agent management environment is used for creating multiple containers emulating distributed environment where peers are active. Java Remote Interface has been used for defining and implementing the bootstrap server.

3.4.1 Agents Communication Implementation

All the agents are developed using FIPA compliant agent framework - JADE. Instead of using RMI or socket based communication between various agents including the mobile agent (RA), agent communication language (ACL) has been used for communication particularly using performative (REQUEST, INFORM, and CLP (Call_For_Proposal)). In addition as the RA is a mobile agent, it is further required to register FIPA standard FIPA_SL0 (slCodec) content language.

Agents do not invoke methods on other agents and communicate through ACLMessages. Hence, to handle messages from various agents and/or various kinds of messages the author has implemented the use of *MessageTemplate*. A receive method takes a message template as a parameter and only returns messages matching that template. This is an important feature that is implemented for successful multi-agent communication system. Figure 3.6 shows the snippet. The behaviour implemented by the LA includes case:

1. Where the RA communicates to the LA to locate resource name whose keywords are semantically similar to user's query
2. Where the LA informs the RA about selected peer GUID using ACL.

The multi agent system has been designed to receive search requests from the users through the IntA. IntA class has a graphical user interface associated with it that takes input parameters - keywords for the query (search terms for a resource). The *minimum support* and the time to live (*number of hops*) parameters have been defaulted in the experimental setup to be 0.00 and 3 hops respectively. Upon invoking the search, the RA is created by the IntA in the method *onGUIEvent()* as shown in Figure 3.7 that has an identifier - (GUID) and the minimum support and number of hops as the parameters.

In addition to creation of the RA, the IntA is also responsible for displaying results sent by the RA throughout its life cycle. As mentioned before in Section 3.4, the FIPA specification implemented by JADE does not allow agents to communicate to each other using method invocation or more specifically in this case remote method invocation, the IntA hence, offers functionality for receiving messages from the RA through ACL implemented in the inner class *ReceiveMessageRecon*. This inner class extends the *CyclicBehaviour*, that creates instance of *MessageTemplate* for only receiving messages sent by instances of the RA created by this instance of IntA using *MatchConversationId("results")* and *MatchPerformative(ACLMessage.INFORM)*. See Figure 3.8 for details of *MessageTemplate* for receiving message from the RA. The IntA also has an inner class *ReceiveTerminationRecon* that extends *SimpleBehaviour* for receiving termination message from the RA when RA has reached end of its life cycle or if a matching resource has been discovered.

The RA is responsible for discovering the resource on other nodes by migrating to those nodes. The node that is most likely to host the resource is provided by the IntA that holds directory of nodes for routing the RA on the overlay network. Again, the communi-

```

protected void onGuiEvent(GuiEvent ev)
{
    command=ev.getType();
    ....
    if(command==NEW_RECON_AGENT)
    {
        jade.wrapper.AgentController a = null;
        try
        {
            Object[] args=new Object[5];
            args[0]=getAID();
            System.out.println(args[0]);
            args[1]=gui.getQuery();//query
            args[2]="0.0";//minimum support
            args[3]=(Object)name;
            args[4]="2";//number of hops
            String name_of_Agent="Reconnaissance_Agent_"+(count++);
            a=home.createNewAgent(name_of_Agent,ReconnaissanceAgent.class.getName(),args);
            a.start();
            agents.add(name_of_Agent);
            gui.activeAgents(agents);
        }catch(Exception ee)
        {
            System.out.println("Problem while creating new agent "+ee);
        }
        return;
    }
}

```

FIGURE 3.7: Creation of the RA in the method *onGuiEvent()* from class InterfaceAgent

```

//inner class
private class ReceiveMessageRecon extends CyclicBehaviour
{
    MatchStore matchStore=null;
    MessageTemplate mt =
    MessageTemplate.and(MessageTemplate.MatchConversationId("results"),MessageTemplate.MatchPerformative(ACLMessage.INFORM));
    public void action()
    {
        try
        {
            ACLMessage reply = receive(mt);
            if(reply!=null)
            {
                matchStore=(MatchStore)reply.getContentObject();
                gui.setResult(matchStore);
            }else
            {
                block();
            }
        }catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}

```

FIGURE 3.8: *ReceiveMessageRecon* class showing *blocked* state of when reply received is *null* and the *MessageTemplate* for receiving messages from the RA

cation between the IntA and the RA is using ACL and the *MessageTemplate* uses *Match-Performative(ACLMessage.REQUEST)*. See InformationAgent.java code in Appendix - C.

Upon migration, the RA communicates to the LA as shown in Figure 3.6 for a matching resource. All the results obtained are communicated back to the IntA through the *MessageTemplate* described above.

3.4.2 Feature Matrix - Frequency-Based Indexing

Feature matrix is created for shared resources hosted by nodes. A modified form of feature matrix called feature vector is used to present node based on the content shared. The process of indexing has two sub tasks. The first subtask is the assignment of tokens for a resource and the second subtask is the assignment of weights to the tokens. The weight is numeric value that is directly proportional to the importance of the token in a resource. The weights are of type integers. These integers present the count of number of unique tokens in a resource. The text for a resource is split into tokens where tokens are only content keywords (adjectives, adverbs, nouns and verbs). The content keywords form index. The representation for a node called feature vector is created by using the indexes for entire collection of resources on that node. Number of feature vectors when collated on bootstrap server form the master feature matrix for entire collection of participating nodes. Each node also has keyword-resource feature matrix that is created by recording frequency of keywords for each resource. The process of locating keywords is given in following pseudocode:

1. Receive the text to be parsed.
2. Build a custom stopword list based on the type of text.
3. Generate a list of tokens from the text of given resource.
4. Initialise a list of content words and loop through the list of tokens.
 - (a) Skip the token if it does not begin with a valid character.
 - (b) Skip tokens that are less than 3 characters long.
 - (c) Skip tokens that are found in the stopword list.
 - (d) Add the token to the list of content words.
5. Return the list of content words.

```

public void getKeywords()
{
    int index=0;
    String keywordSet="";
    ArrayList keywords1 = message.getKeywords(); //Data Structure for holding keywords
    for(int i=0;i<keywords1.size();i++)
    {
        keywordSet+=(String)keywords1.get(i)+" "; //Concatenate Keywords
    }
    //Tokenise keywordSet based on Regular Expression
    StringTokenizer token = new StringTokenizer(keywordSet);
    //Get number of rows
    size=token.countTokens();
    //Create Array based on number of keywords found
    makeTKArray(size);
    //Loop and count keywords
    while(token.hasMoreTokens())
    {
        tk[index]=token.nextToken();
        findTokenFrequency(tk[index]);
    }
    //add keywords and their frequency into TreeMap
    database.addKeywords(map);
}

```

FIGURE 3.9: Method *getKeywords()* for getting keywords and their frequencies and holding in data structure *TreeMap*

For finding frequency of keywords found using the above pseudocode, method *getKeyword()* in private class *FrequencyFinder* is invoked. The method stores all keywords in *TreeMap* data structure as shown in snippet Figure 3.9. *TreeMap* guarantees that the map will be in ascending key order, where keys are distinct keywords and values are the frequency of each key. The list of tokens/keywords in step 3 is stored in *ArrayList* data structure. For loop is used for getting frequency of each token and storing the counting as a value in *TreeMap*. This forms a feature vector for each resource and the collection of feature vectors for all resources on a node form a *keyword-resource* matrix. This functionality is achieved by concatenation of all feature vectors to form a sparse matrix called *masterKeywordMatrix* in private class *LocalDatabase*. The *masterKeywordMatrix* is a two-dimensional array of type *double*. Each node is represented by concatenated list of keywords and their frequency that is globally maintained by in class *MasterList*.

3.4.3 Implementation of Latent Semantic Indexing and Singular Value Decomposition

Frequency-based indexing method cannot utilise any global relationships with the resource collection Konchady (2006). LSI indexing method based on the SVD transforms the *keyword-resource* matrix such that major intrinsic associative patterns in the collection are revealed.

Algorithm 3.1 Algorithm for Latent Semantic Indexing of keyword-resource or keyword-node matrix

Input keyword – resource or keyword – node matrix $A(i, j)$

*$A = (A(i, j))$ where $i = 1, t$, $j = 1, r$ ($t * r$) matrix of keywords and resources*

Perform SVD : $A = USV^T$

Set all but the k highest singular values to 0

Compute $A_k = U_k S_k V_k^T$ by retaining the largest k singular values

Output A_k Latent Semantic Index

```

/**
 * SVD calculation
 */
public void calculateSVD(double[][] matrix)
{
    Matrix mat = new Matrix(matrix);
    SingularValueDecomposition svd = mat.svd();
    U = svd.getU(); // Left Eigen Vectors
    S = svd.getS(); //Singular Values
    S_inverse = S.inverse();
    V = svd.getV(); //Right Eigen Vectors
    V_transpose= V.transpose();
}

```

FIGURE 3.10: Realisation of Singular Value Decomposition from frequency based keyword-resource or keyword node matrix

LSI does not depend on individual keywords to locate a resource, but rather uses concept to find relevant resource. The main purpose of transforming the projection of resource from vector space to LSI space is to locate groupings of resources and use a similar representation for the group (hence a cluster). The algorithm for performing LSI on a group of resources is given as follows (See Algorithm 3.1):

The implementation in the `calculateSVD(double[][] matrix)` method of *MasterList* class the data structure called *Matrix* provided in JAMA API to create a clone of double[][] array and then computes the decomposition of the matrix by invoking method `svd()` (See snippet in Figure 3.10). The return type of this method is *SingularValueDecomposition* that is further used to invoke accessor methods `getU()`, `getS()` and `getV()` for getting left eigen vector (U), singular orthogonal matrix (S) and right eigen vector (V) respectively. The output matrices are then subjected to dimensionality reduction based on top k sigma value in singular matrix (S). The number of sigma values, k , is the floor of the square root of number of resource. A new keyword-resource matrix is generated using the truncated k dimensions.


```

private class KeywordRequestor extends TickerBehaviour
{
    private KeywordRequestor(Agent a)
    {
        super(a,300000); //Timer of 300,000ms
    }

    ...

    //Behaviour Implemented upon expiry of Timer
    public void onTick()
    {
        //Send Message
        ACLMessage request=new ACLMessage(ACLMessage.REQUEST);//Request Message
        request.addReceiver(new AID(nameLA,AID.ISGUID));//Receiver Local Agent
        request.setConversationId("keywords-request");//ID keywords-request
        request.setReplyWith("request"+System.currentTimeMillis());//Update Time
        myAgent.send(request);//Post Message
        ....
        callNodeRegistry();//Update Bootstrap Server
    }
    else
    {
        block();//Blocked State
    }
}
} //end inner class

```

FIGURE 3.11: Index Maintenance task performed recursively by Information Agent

3.4.3.1 Index Maintenance

As the resource collection is dynamic and the nodes are autonomous, new resources and node are added and existing resources and nodes are modified or deleted. The index built from SVD of a keyword-resource or keyword-node matrix is a snapshot of the document collection at some earlier time. The changes made to the collection after the SVD computation, are not reflected in the index. For effective routing, clustering, the index of the bootstrap server must reflect the most recent state of the resource or node collection. Nodes are represented by the content hosted by them and the mobile agent is routed based on most updated state of index. To compensate for these changes, the information agent recursively (after 300,000ms) updates the index by supplying bootstrap server with most recent state of a node. This behaviour is implemented in the inner class - *KeywordRequestor*, that extends *TickerBehaviour* that invokes method *onTick()* recursively after expiry of time passed as parameter in constructor - shown in Figure 3.11.

3.4.4 Similarity Function

The cosine measure is the ratio of sum of the products of common keywords to the products of the lengths of the two vectors. It measures the degree of overlap and uses the presence of keywords to compute similarity. As described in Section 3.2, the author has proposed the use of cosine similarity function for clustering nodes, searching nodes based conceptual similarity between node and query and also for matching the query resource hosted by a node. In following sections, the author presents realisation of these functionalities.

3.4.4.1 Node Learning - Clustering

For the purpose of clustering nodes that are conceptually similar, node represented by keywords is transformed into node vector of k dimensional space on bootstrap server. This transformation is required for comparing node vector to existing other node vectors for calculating cosine similarity. The similarity value of nodes that is less than *minimum_support* constraint provided by user is returned to *InformationAgent*. The realisation of this functionality is provided in *MasterList* class. The return type is serialised object called *Directory* that contains the NodeId, similarity value and shared keywords. The *Directory* data structure forms a local repository and cluster of conceptually similar nodes. The pseudocode for locating nodes belonging to same cluster is as follows:

1. Initialise local node vector based on concatenation of keywords and their weights.
2. Submit local node vector to BootstrapServer.
3. Transform local node vector into k dimensional space.
4. Run a loop until convergence.
 - (a) Calculate cosine similarity $sim(l_k, n_k)$ between the transformed vector and other available node vectors
 - (b) If $(sim(l_k, n_k) > minimum_support)$ then
 - i. Node belongs to the clustered.
 - ii. Add node GUID to *Directory* data structure.
 - iii. Add node's similarity value to *Directory* data structure.
 - iv. Add node's keywords to *Directory* data structure.

- (c) Else if ($\text{sim}(l_k, n_k) < \text{minimum support}$) then
 - i. Node does not belong to cluster, reject node.

5. Terminate when number of nodes converges.

The snippet in Figure 3.12 shows the serialised *Directory* data structure that holds the result of clustered nodes. This object is passed by *BootstrapServer* to *InformationAgent* in order to facilitate the functionality of plotting route upon query for *ReconnaissanceAgent* through overlay network.

```

/**
 * Data Structure for holding the directory peer - keyword matrix used by
 * Information Agent and Bootstrap
 */
public class Directory implements Serializable
{
    //Hold Similarity Value
    double similarityValue;
    //Hold Keyword Frequency Weights
    Matrix keyWeights;
    //Hold Keywords
    ArrayList keywords;
}

```

FIGURE 3.12: Directory data structure used by *BootStrapServer* to pass clustered nodes result to *InformationAgent*

3.4.4.2 Node Searching and Ranking - Content Based Routing

For guided search on an overlay network and hence to reduce saturation, the mobile agent is required to have some heuristics about nodes on the overlay network. As described and implemented in Section 3.4.2, all nodes are represented by the concatenated set of keywords and their respective weights. In order to guide *ReconnaissanceAgent* towards the node that host resource that is conceptually similar to the query passed by user, cosine similarity is measured between the query keywords and the list nodes available to node. Based on *minimum_support*, the selected nodes are sorted and ranked such the node with highest similarity value is ranked as 1. The *ReconnaissanceAgent* is issued with GUID of this selected node that is further used by *ReconnaissanceAgent* to request AMS for container, where the selected node exists. Once the container address is available, the *ReconnaissanceAgent* migrates to this selected node for facilitating query resolving task. The pseudocode for selection and ranking of node is as follows:

1. Assuming that *Directory* containing list of nodes - their similarity values, keywords and GUIDs is available to node.
2. For each node in *Directory*
 - (a) Measure cosine similarity between node and issued query
 - (b) Add node GUID and the similarity value to *HashMap*
3. Sort elements in *HashMap* based on similarity value - to get node with highest similarity value as rank 1
4. While node is not selected
 - (a) If node GUID does not exists in visited nodes array then
 - i. Select node GUID
 - ii. Inform *ReconnaissanceAgent* about GUID of selected node
 - iii. Change state to node selected
 - (b) Else
 - i. Increment index of visited node array.
5. Migrate *ReconnaissanceAgent*

The pseudocode is implemented using a private class *NodeRequestor* that extends *CyclicBehaviour*. Upon receiving an *ACLMMessage.Request* from *ReconnaissanceAgent*, the method checks if the *Directory* is not empty or the list of *clusterNeighbours* exist. All the GUIDs referred to as *keysIPS* are recalled to create a new matrix with their weights including the keywords suggested by user in query. Cosine similarity is calculated and the results are stored in *simR* matrix data structure. *simR* is checked to be valid against user provided *minimum_support* parameter before the chosen node is submitted to *ReconnaissanceAgent* agent. (See snippet in Figure 3.13)

3.4.4.3 Query Resolving

In order to resolve a query - it is represented in k dimensional space like a new resource. The set of query keywords are projected on the existing keywords vector and weighted by the k dimensions. The result of computation is a query vector that can be compared with

```

if(clusterNeighbours!=null)
{
    //GET NODE GUIDs
    Set keysIPS = clusterNeighbours.keySet();
    //LOOP TO FORM WEIGHTED NODE_KEYWORD MATRIX
    for(int j=0;j<keysArray.length;j++)
    {
        ...
        double[][] weightsMatrix = weights.getArray();
        //UPDATE WEIGHTS
        for(int k=0;k<weights.getRowDimension();k++)
        {
            if(weightsMatrix[k][0]==0)
            {
            }else if(weightsMatrix[k][0]>=1)
            {
                for(int u=0;u<weightsMatrix[k][0];u++)
                {
                    updated.add(a.get(k));
                }
            }
        }
        ...
    }
    ...
    //SIMILARITY RESULTS
    Matrix simR=database.getSimMatrix();
    double mins = Double.parseDouble(minSup);
    ...
    //CHECK SIMILARITY VALUE AGAINST MINIMUM SUPPORT
    for(int q=0;q<simRArray.length;q++)
    {
        simVal=simRArray[q][0];
        if(simVal>mins && simVal>temp)
        {
            ...
        }
    }
    //CHOOSE NODE
    ArrayList clientAgents=database.getClientList();
    String chosen = (String)clientAgents.get(indexer);
    System.out.println("THE CHOSEN ONE IS "+chosen);
    //CREATE REPLY TO RECONNAISSANCE AGENT
    ACLMessage reply = messagerec.createReply();
    if(keywords!=null)
    {
        reply.setPerformative(ACLMessage.INFORM);
        reply.setContent(chosen);
    }
    myAgent.send(reply);
}

```

FIGURE 3.13: Realisation of node searching and ranking

Algorithm 3.2 Algorithm for transforming query into k dimensional query vector, calculating similarity and ranking resources

Input

$A_k = U_k S_k V_k^T$ Locally Shared Resources on Node

Query : Q

Perform Q^T

Perform S^{-1}

Compute $Q_k = Q^T U_k S^{-1}$ Transformed Query Vector in k dimensional space

Perform Cosine Similarity Test and Ranking

For $i = 1$ to n

Compute $\text{sim}(Q_k, R_i) = \frac{\langle Q_k, R_i \rangle}{|Q_k| |R_i|}$

Perform ranking

Next

Sort rank based on minimum support

Output

Return $\text{sim}(Q_k, R_i)$, *rank*

other resource vectors in the same k dimensional space. Details of transforming the query to a query vector are provided in Section 3.2.1. Query resolving is performed by *LocalAgent* upon request from *ReconnaissanceAgent*, when mobile agent visits a node. The algorithm for transforming the query into a query vector in k dimensional space is shown in Figure 3.2.

The implementation of algorithm (Algorithm 3.2) is realised through the method *calculateSim()* of private class *LocalDatabase*, used for computing cosine similarity between the query vector and the keyword-resource matrix. The method returns the matrix data structure that contains cosine similarities values for all local resources on a node as compared to query vector. The method computes the numerator that is the sum of product of common keywords. The denominator is computed by products of length of each vector. The ratio is stored in a matrix *simM* and returned to mobile agent RA.

3.4.5 Mobile Agent - Reconnaissance Agent

ReconnaissanceAgent is a mobile agent that is responsible for discovering resources on the overlay network. It is also responsible for migrating from node to node while comparing the

search query against hosted resources. Important methods that have been implemented to realise the responsibilities include:

1. *takeDown()*: This method is overridden and implements *doDelete* method for terminating ReconnaissanceAgent.
2. *afterMove()*: This method is overridden and is responsible for finding local LocalAgent and compare the query against the catalog it is keeping. This method is responsible for the following tasks. a.) if any of the results are greater than minimum support, it is responsible for sending *ACLMessage* to its creator (InterfaceAgent) informing about the discovery - name of file and name of LocalAgent hosting it. b.) checks, if it has made number of hops less than maximum number of hops allowed. If the number of hops are less than maximum allowed then it should communicate to local InformationAgent on this node and get the next migration address and container else it kills itself.
3. *commForJump()*: This method implements the steps that required to be performed by ReconnaissanceAgent before migration to new node.
4. *sendRequest()*: This method sends message to AMS for location of the named static agent (InformationAgent, LocalAgent or InterfaceAgent).
5. *setup()*: This method is an overridden and is responsible for getting parameters for ReconnaissanceAgent.
6. *getNode()*: This method is responsible for communication of ReconnaissanceAgent with InformationAgent to get new node where it should migrate in order to perform resource discovery in case number of hops are lesser than maximum number of hops allowed.

This section includes details of realisation of features mentioned in contributions and objectives in Chapter 1. The author has presented algorithms, pseudocodes and implementation details of these features. In addition, the author also presented details of methods implemented by mobile agent in order to realise its functionality.

3.5 Discussion

In flooding-based systems, upon receiving a query, each peer sends a list of all matching resources to the originating node. This results in increase of load on each node that is linearly proportional to the total number of queries. It must be noted that this load will increase with growth in system size making flooding based approach clearly not scalable. To make unscalable systems scalable literature presents DHT-based system that has limitation of search performance because of rigid *key-value* pairing for propagating the query to resource Chawathe et al. (2003). In the proposed system, routing of RA is heuristic based that provides flexible search semantics based on *keyword-node* pairs and supports attaching keywords to shared resources and content-based similarity retrieval thus making it more scalable. Scalability can also be attributed to the proposed resource discovery mechanism that supports exact and similarity search based on *keyword-resource* matrix unlike flooding-based or DHT-based techniques. The author believes that the proposed system provides the necessary flexibility and performance for effective use of LSI for searching and routing on overlay networks.

Furthermore, it must be noted that this implementation has been realised keeping intra-platform mobility in context. In case of inter-platform mobility - the GUIDs will be undermined as container numbers are not unique across multiple platforms. In such case, the author suggests the use of IP address concatenated with agent type and container id to create a globally unique identifier for an agent at global level.

3.6 Summary of the Chapter

The chapter discussed in detail all the design features that implement the characteristics of resource discovery system as understood and informed in Section 2.3. Details of agent communication that include *MessageTemplate* and *MatchPerformative* are described in Section 3.4.1. Furthermore, description of various features, their implementation and the required algorithms have been discussed in Section 3.4. The details about extensive coding have been removed from main report and added to program listings for readers (See Appendix - C).

In next chapter, experimentation is conducted to test the efficiency and effectiveness of Affinity. Also, included in next chapter are tests that compare results from proposed system to current research works. In addition, evaluation of results is provided in detail in

following chapter.

CHAPTER 4

EXPERIMENTS, RESULTS, AND EVALUATION

The experiments were conducted to evaluate the effectiveness of proposed method for resource discovery using mobile agents. The experiment is bifurcated into two parts. Part-1 investigates to find out the response time (in secs) that it takes to locate a resource (multiple keywords based query) on an overlay network using RA in MAS as compared to flooding. Part-2 investigates the benefit of using RA for informed search based on LSI as opposed to flooding and other routing algorithm inspired by AntHill (Babaoglu et al. (2002); Babaoglu & Jelasity (2008)) and structured P2P systems by (Dasgupta (2003); Kambayashi & Harada (2009)) by finding out the amount of messages that are on an overlay network.

4.1 Design of Experiments

The design of experiments has been setup in order to compare the proposed technique for content-based resource discovery in terms of heuristic search and search performance. The benchmarks are provided by flooding technique and by term-matching, Jaccard coefficient techniques Chawathe et al. (2003); Crespo & Garcia-Molina (2004); Zhu & Hu (2007); Dasgupta (2008); Kambayashi & Harada (2009). Flooding technique was used as benchmark; as it is widely accepted technique and has been used as backbone for purpose of routing and searching in number of resource discovery techniques including the contemporary techniques as proposed by Dasgupta et al. Dasgupta (2003, 2008). Furthermore, as Dasgupta et al. is using this technique for routing in context to MAS, it becomes all the more important to prove the effectiveness in terms of routing and searching of proposed technique in this context. More contemporary researches from Zhu et al. and Kambayashi et al. have proposed the usage of semantics links based on term-based matching or Jaccard coefficient

for resource discovery. Kambayashi et al. uses mobile agent to traverse through overlay network and their technique of preference for matching resources is logical similarity based in Jaccard coefficient Kambayashi & Harada (2009). Kambayashi et al. further uses DHT based structured overlay for migration of mobile agent. Similar approaches has been used in different flavour however (for instance, using DHT for locating nodes, using flooding for routing mobile agent or using term matching for locating relevant results) have been used by many contemporary research works. As Kambayashi et al. is using number of techniques in their approach, the author believes comparing results of proposed method to their technique would provide comparison and evaluation on high degree of intersection of attributes and techniques and a good benchmark. The experiments conducted measure the performance of the proposed method on the basis of following parameters:

- the response time test
- the effectiveness of search technique
- relevance of results
- degree of similarity

4.1.1 Experiment Environment and Test Bed

For comparison to flooding technique as employed by Gnutella, the experimental setup used the open source Java API, JTellav0.7 McCrary & Waters (2000); Forum (2002); Chawathe et al. (2003). This API can be used to create a P2P overlay network and is well documentation on the libraries as well as source code in Java. The setup included 4 peers where 3 peers hosted resources and fourth peer is used for searching resources. Details of each peer including hardware specifications, operating system, IP addresses, number of resources and types of resources is shown in table 4.1.

As seen in table 4.2 total number of nodes participating in Affinity were 10. For the purpose of consistency with benchmark, 4 computers participated in this experiment. In this setup, computer 1 hosted Bootstrap server and 3 computers participated in P2P overlay network. Between these 3 computers, 10 nodes were created, where computer 1 hosted 4 containers hence 4 nodes, computer 2 hosted 3 containers hence 3 nodes and finally computer 3 hosted 3 containers hence 3 nodes. Each container simulated as different node participating in P2P overlay network. The hardware specification of machines is as provided

Peer Name	Peer 1	Peer 2	Peer 3
Operating System	Microsoft Windows Vista Home Premium	Microsoft Windows 7 Home Premium	Microsoft Windows XP Professional Service Pack 2
Processor	AMD Athlon Dual Core QL-62 2.00 GHz	Celeron (R) Dual Core CPU T3000 @ 1.80 GHz	Intel Pentium 4 @ 2.50 GHz
RAM	3 GB	3 GB	512 MB
IP Address	192.168.1.2	192.168.1.5	192.168.1.7
Number of Resources Shared	13	8	8
Type of Resources	8 pdf files 3 docx files 2 doc files	6 pdf files 2 rar files	8 pdf files

TABLE 4.1: Gnutella flooding peers test bed

Specification	Value
Total Number of Computers	4
Bootstrap Server	1
Computers Participating	3
Number of Nodes Participating	10
Maximum Number of Hops	2
Total Number of Shared Resources	27
Minimum Support	0.0

TABLE 4.2: MAS test bed

in table 4.1. Further specifications regarding MAS and keywords for resources shared are shown in table 4.2 and table 4.3.

Local Agent	Keywords Shared
LA1	sun moon earth mars mercury venus
LA2	moon pluto
LA3	sun stars one two
LA4	one two three four five six mars
LA5	moon
LA6	jupiter saturn neptune pluto
LA7	moon saturn pluto
LA8	two neptune
LA9	pluto earth one
LA10	one sun two moon

TABLE 4.3: Keywords used for sharing resource on each node

The objective in test 2 is to compare the effectiveness of indexing technique, relevance of results and degree of similarity. The experiments in test 2 used a MEDLINE data set that consisted of 1033 documents University (1999). After removing of stopwords and filtering of nouns, verbs, adjectives, and adverbs, 5735 indexing terms (lexicons) were found. The details of data set can be found on media disc. This data set was used specifically as

all resources have already be categorised and attributed with features such as relevance and similarity. The objective was to find out if the proposed technique provides similar results and then to compare the results with techniques used by other research works. Hence, the prepared results served as benchmark for comparing the effectiveness of proposed technique to other relevant works.

4.2 Test 1 - Comparison to Flooding Technique

4.2.1 Experiment 1 - Response Time and Evaluation

The objective of experiment 1 was to calculate the response time for query on an overlay network. The performance metric response time is defined as the time elapsed between a user initiating a request and receiving the results. This includes the time taken for agent creation, time taken to visit the node and the processing time to extract the required information. Once the response time is available it can be concluded that which method is more effective with respect to amount of time it takes to locate a resource on an overlay network. It is observed from the bell curve that amount of time it takes to find a particular resource in proposed method is consistent and ranges between 5s to 6s. Flooding however does not have any consistency in response time. It is observed from bell curve shown in Figure 4.1 that response time can vary from few seconds to few minutes. Furthermore, it is observed that in flooding 14 queries out of potential 28 queries has response time of $< 5s$ which approximates to 42% of total number of queries, where when using proposed method the author observed that 67% of queries were replied with resource location in $< 5s$ and 23% of queries replied in $< 6s$.

It is evaluated that overall response time, when using the proposed method is lesser than the case of Gnutella using flooding technique. But it should be noted that a lower response time does not measure the effectiveness of search technique in terms of successful results as described further in Section 4.2.2. The author concludes that lower response time is attributed to mainly two reasons. Firstly, as Gnutella is pure P2P network, it is required of participating peers to communicate their status using PING and PONG messages on the overlay network. This results in high amount of traffic on overlay network and results in saturation. It is observed, as mentioned in Section 4.2.3, that PING and PONG activity together amount to 97.5% of messages. This results in latency and hence low response time. Secondly, as resources to be located are searched based in multiple keywords do not always

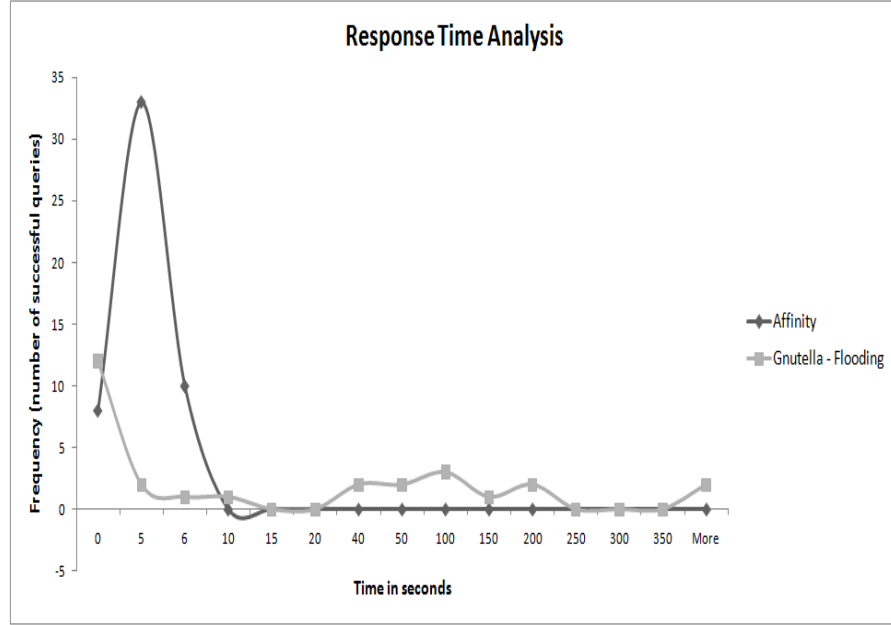


FIGURE 4.1: Frequency distribution of response time analysis - Gnutella vs. Affinity

match the file name of resource to be located, it amount to low response time as query may not match the resource completely.

4.2.2 Experiment 2 - Effectiveness of Search Technique and Evaluation

In experiment 2, the objective was to investigate the effectiveness of search using proposed method as compared to flooding. For achieving this objective, the experiment setup was to compare successful queries to unsuccessful queries. It was realised through experiment using proposed method that out of 30 queries, 24 responded with query hit, 4 queries did not have any response, and 2 queries replied as NaN network (See Figure 4.2).

Furthermore, it was realised that NaN is due to explicit specification of *minimum support* parameter as 0.0. The nodes in similarity with 0.0 did not host the content that was required by user. In case, of flooding, 26 queries were passed through various nodes. 32% of queries had query hit and 68% of queries failed (See Figure 4.3).

4.2.3 Observations

Dasgupta (2003); Kambayashi & Harada (2009) has confirmed that no matter how many peers or resources are there on an overlay network, the flooding technique generates a con-

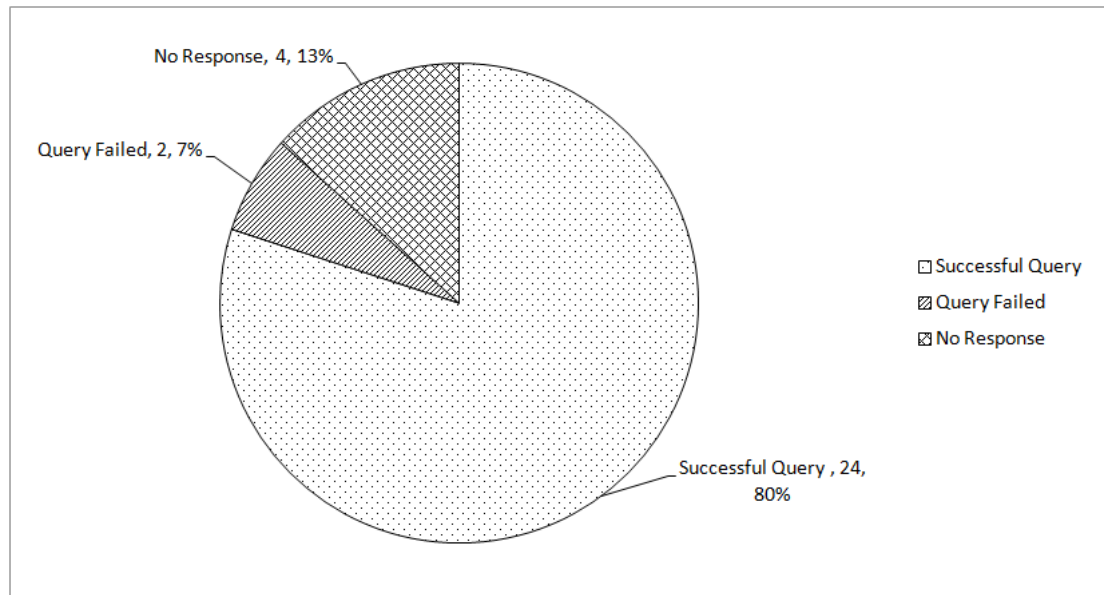


FIGURE 4.2: Query successful vs. unsuccessful - Affinity method

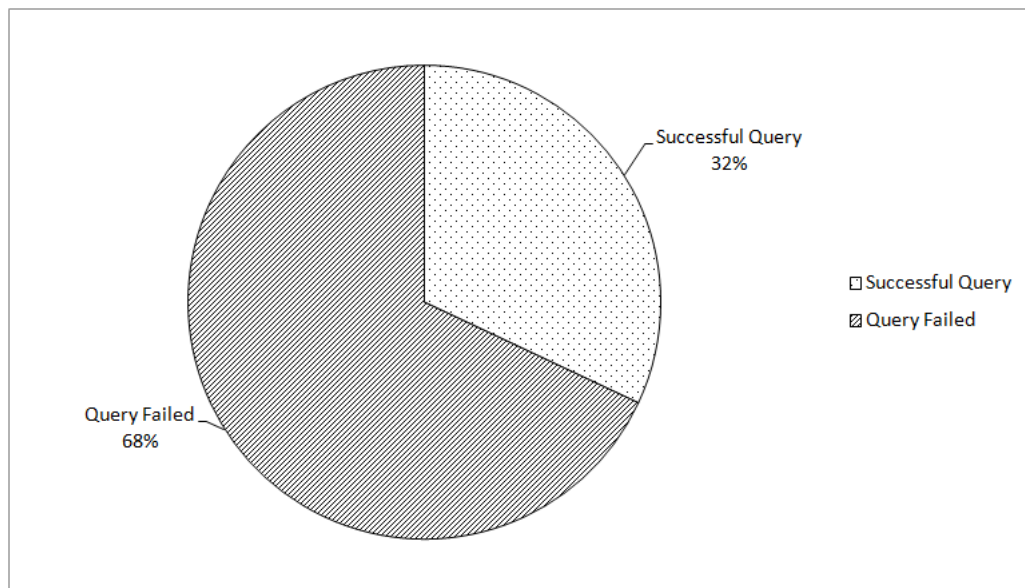


FIGURE 4.3: Query successful vs. unsuccessful - flooding method

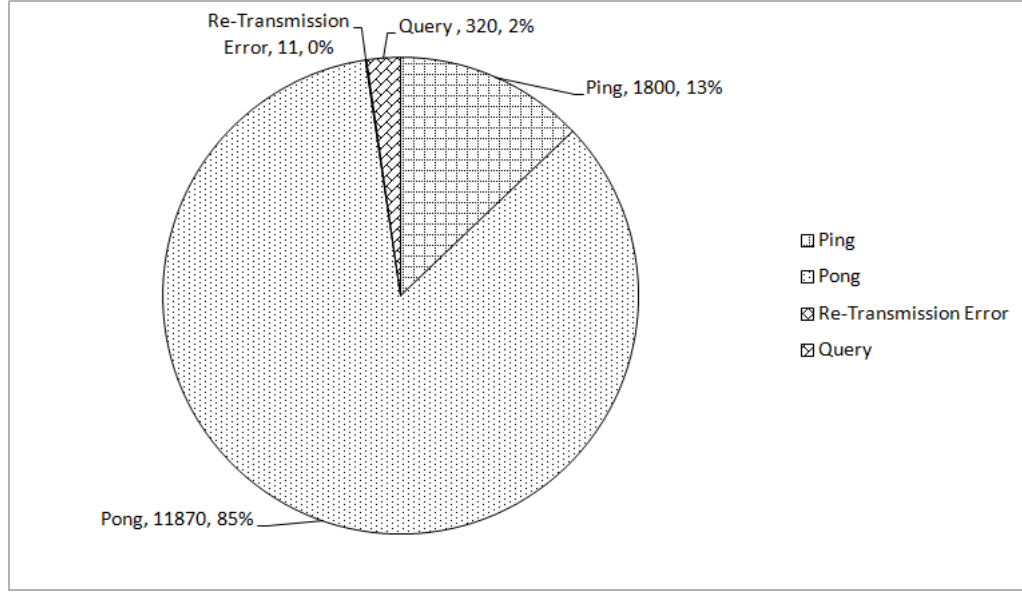


FIGURE 4.4: Division of packets for Gnutella

sistent number of messages on an overlay. The author has observed in flooding that the amount of traffic or messages on an overlay network or even response time increase or decrease is attributed to mainly the PING and PONG activity. This continuous stream of messages is produced by the peers to check existence and current status of other peers. The author used Wireshark to monitor the Gnutella packets Wireshark (2010). The screenshot in Figure shows Gnutella packets upon filtering. A total of 14008 Gnutella packets were analysed when overlay network was subjected to 2 queries. It is clear from the pie chart (Figure 4.4) that 84.7% of traffic is related to PONG descriptors, 12.8% to PING, 2.22% to QUERY, and 0.07% to re-transmission errors. Gnutella connections are relatively unstable, which lead the nodes in iterative effort for discovering other nodes on overlay network as opposed to nodes joining and leaving network autonomously.

It is also observed from the graph in (Kambayashi & Harada (2009)) that no matter what is the number of resources shared, as long as number of peers is constant the number of messages (bytes) will stay constant.

4.2.4 Critical Analysis

However, this raises another issue of why there is a decrease in number of messages also claimed by Dasgupta (2003); Kambayashi & Harada (2007, 2009). The author observed and

No.	Time	Source	Destination	Protocol	Info
13	2.024801	192.168.1.2	192.168.1.5	Gnutella	Ping
14	2.078781	192.168.1.5	192.168.1.2	Gnutella	Pong
15	2.079368	192.168.1.5	192.168.1.2	Gnutella	Pong
16	2.084965	192.168.1.2	192.168.1.5	Gnutella	Ping
17	2.143363	192.168.1.5	192.168.1.2	Gnutella	Pong
45	7.024362	192.168.1.2	192.168.1.5	Gnutella	Ping
46	7.064915	192.168.1.5	192.168.1.2	Gnutella	Pong
47	7.069004	192.168.1.2	192.168.1.5	Gnutella	Pong
48	7.071905	192.168.1.5	192.168.1.2	Gnutella	Pong
50	7.073652	192.168.1.5	192.168.1.2	Gnutella	Pong
72	12.024481	192.168.1.2	192.168.1.5	Gnutella	Ping
73	12.070033	192.168.1.5	192.168.1.2	Gnutella	Pong
74	12.070674	192.168.1.5	192.168.1.2	Gnutella	Pong
75	12.076305	192.168.1.2	192.168.1.5	Gnutella	Ping
76	12.132211	192.168.1.5	192.168.1.2	Gnutella	Pong
95	17.024432	192.168.1.2	192.168.1.5	Gnutella	Ping
96	17.045163	192.168.1.5	192.168.1.2	Gnutella	Pong
97	17.045774	192.168.1.5	192.168.1.2	Gnutella	Pong
98	17.051509	192.168.1.2	192.168.1.5	Gnutella	Ping
99	17.077990	192.168.1.5	192.168.1.2	Gnutella	Pong
117	18.942526	192.168.1.5	192.168.1.2	Gnutella	Pong
119	18.942777	192.168.1.5	192.168.1.2	Gnutella	Pong
125	22.024428	192.168.1.2	192.168.1.5	Gnutella	Ping
126	22.061713	192.168.1.5	192.168.1.2	Gnutella	Pong
127	22.071773	192.168.1.2	192.168.1.5	Gnutella	Ping
128	22.072472	192.168.1.5	192.168.1.2	Gnutella	Pong
129	22.078159	192.168.1.5	192.168.1.2	Gnutella	Pong
149	27.024463	192.168.1.2	192.168.1.5	Gnutella	Ping
150	27.070655	192.168.1.5	192.168.1.2	Gnutella	Pong
151	27.071244	192.168.1.5	192.168.1.2	Gnutella	Pong
152	27.075368	192.168.1.2	192.168.1.5	Gnutella	Ping
153	27.099725	192.168.1.5	192.168.1.2	Gnutella	Pong

Frame 13 (77 bytes on wire, 77 bytes captured)
 Ethernet II, Src: HonHaiPr_78:1b:8f (00:23:4e:78:1b:8f), Dst: LiteonTe_78:63:9f (70:1a:04:78:63:9f)
 Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 192.168.1.5 (192.168.1.5)
 Transmission Control Protocol, Src Port: 60531 (60531), Dst Port: gnutella-svc (6346), Seq: 1, Ack: 1, Len: 23
 Gnutella Ping

0000 70 1a 04 78 63 9f 00 23 4e 78 1b 8f 08 00 45 00 p..xc..# NX....E.
 0010 00 3f 75 45 40 00 80 06 02 1c c0 a8 01 02 c0 a8 ?uE@.....
 0020 01 05 bc 73 18 ca a3 fd 88 d0 d3 be fd df 50 18 ...S....P.
 0030 0f c2 f5 0a 00 00 00 00 00 00 00 01 bd 00 00
 0040 01 25 31 6e ef 94 00 01 00 00 00 00 00%ln.....

FIGURE 4.5: Gnutella packets analysed using Wireshark

evaluated that the decrease in number of messages in these multi-agent systems is due to decrease in number of hops to locate a resource. Kambayashi & Harada (2009), claims that number of messages on overlay network will decrease with increase in number of resources. This is because the overlay network has become more resourceful and hence almost all peers have links to other peers, which means that when the SA enquires from directory services on NA about peer to migrate to, it is capable of informing SA about the highest possible logical distance value because of its resourcefulness. This is observed in proposed method too and the author agrees with Dasgupta (2003); Kambayashi & Harada (2009). It is evaluated in Section 4.3.1, that number of inter cluster links are on average higher than case where, logical distance value was used to create semantic links between nodes. More number of links makes the overlay network more resourceful thus reducing number of hops and reducing number of messages on network. Furthermore, it is evaluated through precision-recall results where the author defines precision as the ratio of number of relevant resources/nodes found during search to number of search results and recall as the ratio of relevant resources/nodes found to total number of relevant resources/nodes in corpus. Though, number of inter-cluster links are higher that may result in compromise of precision, however, we achieve higher

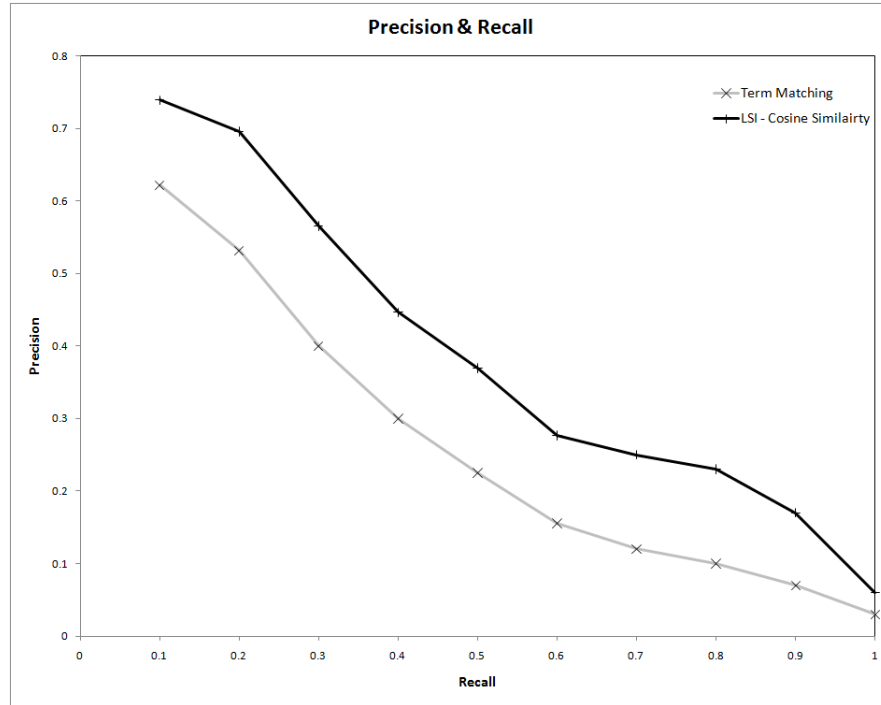


FIGURE 4.6: Precision and recall results comparing LSI to TF-IDF indexing model

recall making degree of relevance higher (See Figure 4.6) due to efficient indexing technique. Together, with results from reduced response time, higher recall and greater number of successful queries it can be concluded, that lesser number of messages exist on network.

In test 2 - Section 4.3, the author investigates the effectiveness of their techniques/algorithms to reduce number of messages and compare them to proposed method.

4.3 Test 2 - Comparison to Other Routing Techniques/Algorithms

The aim of this test is to investigate the effectiveness of the routing mechanism employed by Kambayashi et al. that calculates the logical distance between the nodes based on the resources shared by that node as compared to LSI based clustering of nodes and routing based on calculation of cosine similarity between search query and the lexicons shared by nodes. This experiment also indirectly studies the effect on amount of message on overlay network. Replicating exact environment as used by Kambayashi et al. has been a tedious process as they are using Overlay Weaver and Agent Space both tools developed by them and changed to accommodate messaging between agents through Overlay Weaver Kambayashi

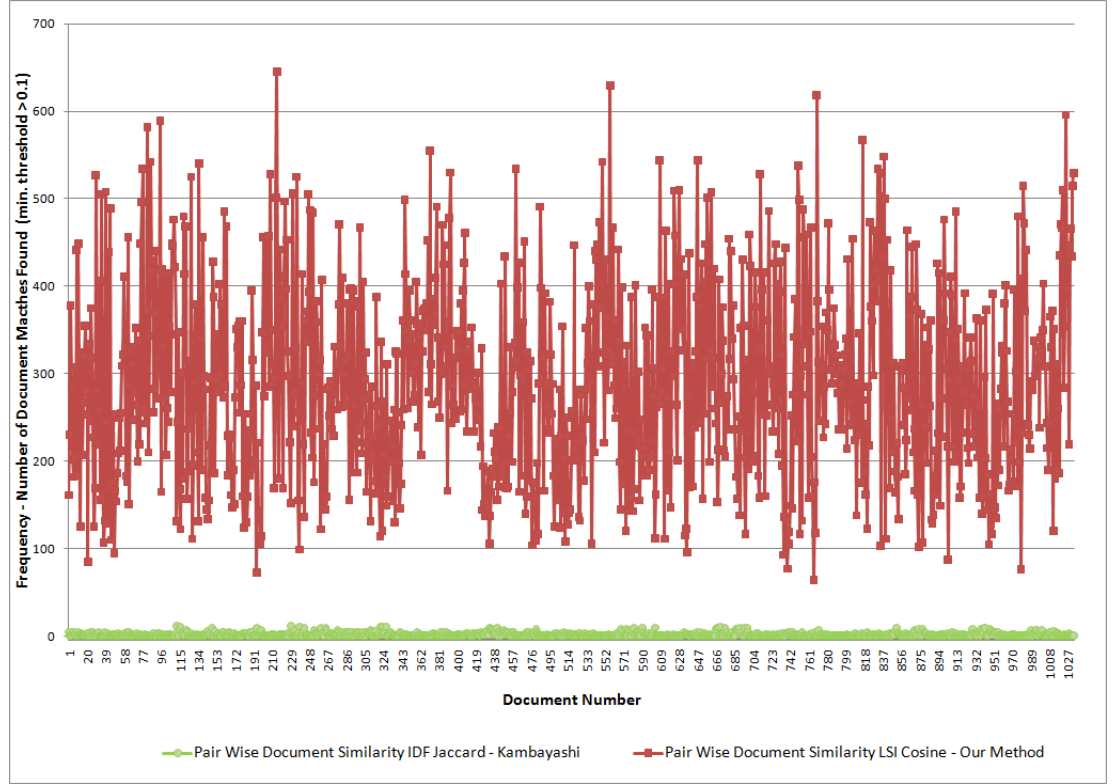


FIGURE 4.7: Pair-wise document similarity TF-IDF Jaccard vs. LSI Cosine

& Harada (2009).

Though, the author evaluated their results and observed that the decrease in number of messages is due to increased similarity score (Jaccard Similarity) between shared lexicons. In this context, the author designed another experiment that would compare their indexing and routing algorithm to the proposed method by comparing its effectiveness on third party data provided by University (1999). The effectiveness was evaluated in different experiments.

4.3.1 Experiment 1 - Pair-Wise Document Similarity And Evaluation

In experiment 1, pair-wise document similarity is investigated by comparing Jaccard similarity (subset used by Kambayashi & Harada (2009)) and Cosine similarity (used by proposed method Singh et al. (2009)). In case of Kambayashi et al., the test required normalising the term-document matrix using term-frequency and inverse document frequency indexing (TF-IDF) for measuring Jaccard similarity Kambayashi & Harada (2009). In proposed case,

the test required creating the normalised latent-semantic indexed matrix for measuring Cosine similarity as described in Section 3.2.1. The effectiveness in experiment 1 is studied by finding out number of documents that match where the minimum threshold is > 0.1 . The result of number of document will indicate the resourcefulness of overlay network, as that is used to cluster the nodes. In other words, more is the number of matched documents, larger is the cluster, and more are the chance for mobile agent to locate a resource which would mean lesser number of migrations for mobile agent and hence, less number of messages on overlay network. The author, evaluated from the following graph (Figure 4.7) that using LSI and cosine similarity, clearly has larger number of pair-wise matches, between documents and hence, provide larger cluster and links between clusters.

It is evaluated that larger is a set of similar documents, more resourceful is the overlay network, hence lesser number of hops are require by RA to locate a resource. The pair-wise documents similarity is large in case LSI technique used in proposed method, hence number of messages required by RA to locate a resource will be lesser and in this case much lesser than flooding (Aberer et al. (2004); Chawathe et al. (2003)) and logical distance method (Dasgupta (2003); Kambayashi & Harada (2007, 2009)) making proposed method for routing RA through overlay network more efficient in terms of time and bandwidth consumption.

4.3.2 Experiment 2 - Effectiveness of Search Technique And Evaluation

In experiment 2, the aim was to investigate number of documents that found to be similar in to search query. Large number of documents effectively indicate:

1. Large number of nodes for the RA to migrate to for locating resources
2. Better inter-cluster link for routing the RA through overlay network.

It is highly important that mobile agent can traverse through overlay network for locating the resource.

If routing links cannot be established between clusters - it would indicate:

1. Mobile agent cannot locate a resource because of its incapability to migrate to different clusters or
2. Mobile agent will provide results that are less precise.

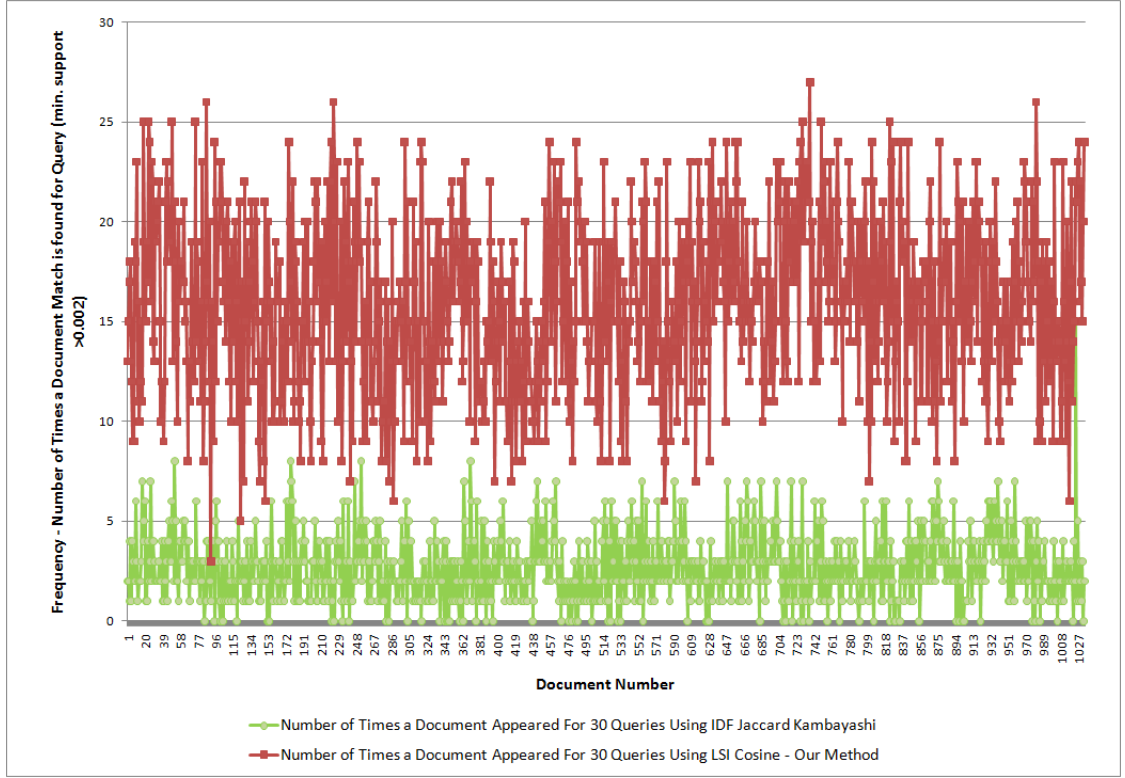


FIGURE 4.8: Number of times a document appears for 30 queries Jaccard similarity vs. Cosine similarity

It must be noted that larger number of matches also mean large number of nodes to be visited by the RA hence more number of message on overlay network which in effect means higher bandwidth consumption. This however is controlled in proposed case by introduction of factor called *minimum support* as mentioned in Section 3.2.1, that is set by user to reduce the number of selected nodes for the RA to visit. The author conducted similarity test on corpus of 1033 documents by subjecting them to 30 different queries University (1999). The following graph (Figure 4.8) was obtained as a result of this experiment, informing number times matched documents is found for 30 queries where *minimum support* is > 0.002 .

It is observed that proposed method is resulting in larger inter cluster links and also large number of nodes where the RA can potentially visit as compared to logical distance method used by Kambayashi & Harada (2009).

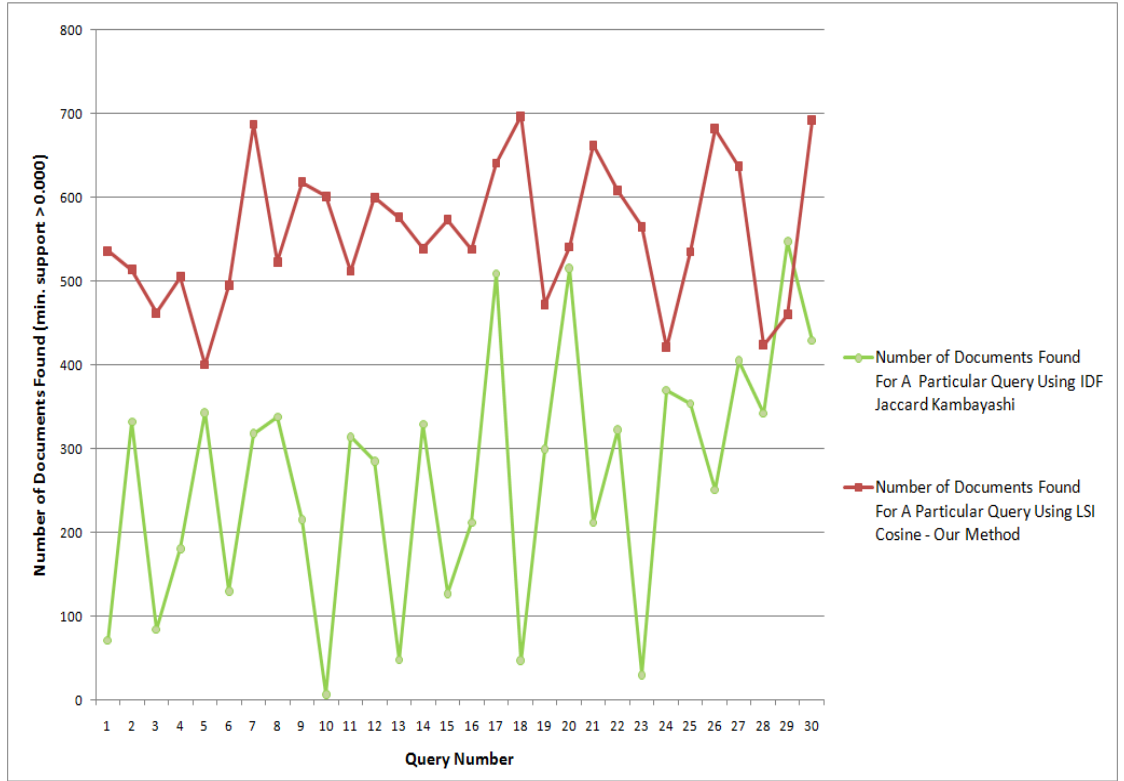


FIGURE 4.9: Number of documents found for 30 separate queries on corpus of documents

4.3.3 Experiment 3 - Effectiveness to Locate Resources and Evaluation

In experiment 3, the aim was to investigate effectiveness of proposed method to locate the resource. Keeping that in context, in general terms it means - number documents found per query using proposed method as compared to the logical distance method. Similar to experiment 2, for achieving the aims of this test, the document corpus was subjected to 30 queries and number of documents found per query were obtained for minimum support > 0 . This number was compared for LSI based Cosine similarity and TF-IDF based Jaccard similarity. The graph (Figure 4.9) shows that number of documents using LSI Cosine method used in proposed method is higher than TF-IDF Jaccard method. It is further evaluated, that a larger number of documents associated with a query means 1. higher cluster links 2. larger set of relevant documents found as part of resource discovery. Of course, as mentioned in experiment 2, larger set of documents can also indicate irrelevant information, but this can be capped using parameter *minimum support* as mentioned in section 3.

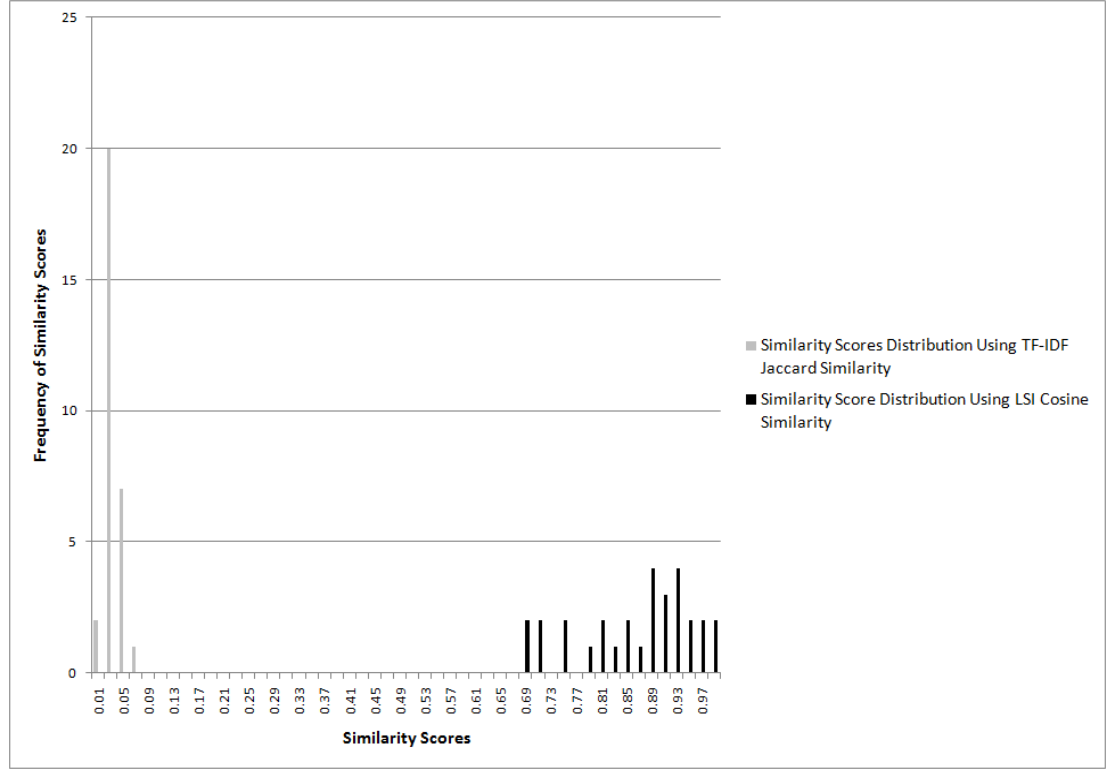


FIGURE 4.10: Similarity score distribution TF-IDF Jaccard vs. LSI Cosine

4.3.4 Experiment 4 - Degree of Relevance of Results and Evaluation

In experiment 4, the aim was to investigate the degree of relevance of results obtained during search process by the RA. Again, similar to experiment 2, the corpus was subjected to 30 queries to find out about similarity scores. The highest similarity score obtained is assumed as resource that is best match to a given query. The objective was to collate the highest similarity scores and find their frequency distribution. This process would:

1. offer insights into relevance of results
2. inform which method is capable of extracting best match documents.

Perhaps, if the same document is found a result of search, using both methods, if logical distance is low, it may safely be assumed that mobile agent may take more time or even more number of hops to reach the node.

It is observed from the graph (Figure 4.10), that using proposed method the similarity scores tend to be on higher end of frequency distribution as opposed to other research

works. This indicates that it is of utmost importance the similarity scores are high which would effectively mean fewer messages on overlay network and better response time.

4.4 Discussion

In order to test the proposed method for content-based routing of mobile agents using LSI on large scale network, the literature offers only a few simulation environments.

A simulator called Swarm is a general purpose software package for simulating, distributed artificial worlds written in Tcl Lingnau & Drobnik (1999). It is particularly useful for large number of autonomous entities (“agents” – not to be confused with “mobile agents”) with an environment. Using this, the global and adaptive behaviour of the proposed system can be studied Lingnau & Drobnik (1999). Anslem Lingnau et al. Through their research offer an extension to Swarm system by including infrastructure for mobile agents. Their extension, allow mobile agent collaboration other agents and also allowing for computations and migrations Lingnau & Drobnik (1999). Included, in this simulator are some routing techniques for studying network load and response time for agent to complete a given task. This environment is suitable for simulating the proposed technique, as long as it allows new routing techniques to be added. One of the drawbacks of this environment is their non standard use of messaging techniques by use of invocation of remote methods rather than standard agent communication language. This will prevent accurate results with regard to response time and efficiency of network usage.

Another simulator that is written in Java and has been used by some researchers for agent-based simulations is Repast North et al. (2006). Due to object-oriented nature of the underlying programming language, it supports computational elements that make agent autonomous (an important characteristic required for agents) Bandini et al. (2009). Furthermore, object oriented nature of Java offers encapsulation of state, actions and action choice mechanism in agent’s class. It also simplifies integration of external APIs such as JADE in this case. This simulation platform does not specifically support realisation of agents and interaction models as standardised by FIPA.

AgentSim developed by IBM has been used by researchers as simulator for simulation of agents Trillo et al. (2007). The simulator is library built only for Aglet - agent development platform Trillo et al. (2007). As mentioned in Chapter 2, Aglet does not support ACL, instead only offer synchronous remote method invocations that are not favourable

for simulating proposed technique Trillo et al. (2007).

Chen et al. developed Mobile-C that conforms to the FIPA standards both at agent and platform level. It also extends FIPA standard to support mobile agent protocol to direct agent migration process. Agent migration is achieved through FIPA-ACL messages encoded in XML Bo Chen (2006). FIPA ACL is effective way for inter-platform agent migration in FIPA compliant Agent systems as both agent communication and migration share the same communication mechanism. The development of simulator is done in C or C++ which makes inter-language barrier for communication, as developments have been done in Java and JADE API. However, the author believes that using CORBA, for inter-language communication can be conceived for successful simulation. This may require extensive writing of interfaces for the developed system and various computational models.

The author understands the issue to test scalability of system on large scale network is important, which will be created as part of simulation in further work. The author believes that Mobile-C offers promising simulation platform for simulating the proposed system on large scale network.

4.5 Summary of the Chapter

In this chapter, the author has provided details of various experiments, that were conducted and describe the characteristics of the resource discovery system, using mobile agent - Affinity as well as provide insights into one-on-one comparison with other routing techniques used in older and current research works. The author also discussed the choice of simulators, their features and drawbacks for large scale mobile agent based simulation.

In next chapter, the author provides discussions on concepts provided by researchers and compare them to concept listed by proposed work by benchmarking characteristics of P2P and resource discovery systems using mobile agent.

CHAPTER 5

DISCUSSION, CONCLUSIONS, AND FUTURE WORK

This chapter discusses other related research works and critically analyse the concepts presented by them. The results obtained as part of experimentation in Chapter 4 are promising and the author believes further discussion of the concepts presented by some of the related works as collated in literature survey is useful for readers.

The works done by Zhu et al ESS, Dasgupta et al, Kambayashi et al, and Crespo et al are related to this research work for development of P2P system for resource discovery and the first sections of chapter provides related discussions Dasgupta (2003); Crespo & Garcia-Molina (2004); Zhu & Hu (2007); Dasgupta (2008); Kambayashi & Harada (2009). In ending sections, the author has collated the future works, that can be undertaken and can be potentially useful with this research work in context. Also, the conclusions have been provided.

5.1 Discussions - Analysis of Other Research Works

In this section, the author discusses related works similar to conducted research work undertaken.

Crespo initially presented the idea of routing indices for controlling the amount of flooding and saturation of overlay network Crespo & Garcia-Molina (2002). The concept however suffered from maintenance of distributed-index on various nodes that itself generated it own large amount of traffic.

Later, Crespo *et al.* introduced the idea of semantic overlay networks (though not in a P2P context) where the nodes can be clustered to form an overlay network Crespo & Garcia-Molina (2004). Crespo *et al.* use explicit term semantics to building routing indices

Crespo & Garcia-Molina (2004). They assign documents with terms indicating related realms, and maintain in each peer a statistic table containing term-based routing indices, which indicates how many documents would be found, if probes the query of that term to a neighbour peer Liu et al. (2004).

The author, understands that Crespo et al brought improvement to searching but as most latent semantics analysis proved, only terms-based statistics cannot fully capture resource characteristics as terms also have underlying correlations and semantics Deerwester et al. (1990); Liu et al. (2004). The author has been inspired from the idea to form relationship between nodes but proposed system uses these relationships for coordination of resources that are managed by nodes and further use it for informed routing of the RA.

Zhu *et al.* presented the use of information retrieval from unstructured and structured P2P system by use of semantic links between the nodes Zhu & Hu (2007). The query flooding on P2P network is controlled using routing based on Jaccard similarity technique. However, as described in tests the results obtained from normalised LSI based cosine similarity technique are far superior on terms of number of document matches and higher similarity scores. Furthermore, their system is not a mobile agent based resource discovery system which as mentioned in literature greatly improves upon the classical unstructured and structured P2P system. Proposed work contributes towards the dynamic organisation overlay network based on resources published by nodes. The relationship between nodes and resources for guidance of agent (direction) on overlay network is central and crucial.

Dasgupta *et al.* (Dasgupta (2003, 2008)) research work is greatly inspired from Babaoglu et al work on Anthill in Babaoglu et al. (2002); Babaoglu & Jelasity (2008). The author here presents analysis of Dasgupta's research work as they have used MAS.

Dasgupta *et al.* introduced the used of mobile agents for P2P resource discovery Dasgupta (2003, 2008). Their system is based on referrals made by search agents. Clearly, in their system the behaviour of search agents evolve and get better, based on the trails established by searches done before. In contrast to proposed work, they do not use the routing tables for guiding the search agent through the overlay network as done in proposed work using directory facility made during initial registration of peer on bootstrap server. Furthermore, they did not introduce the use of peer-keyword semantics to form clusters of semantically similar peers. Clearly, they are using the classical technique of flooding to discovery resources that improves over time based on the search trails left by previous searches.

Kambayashi *et al.* has provided method of resource discovery by using mobile agent and DHT Kambayashi & Harada (2009). Like proposed work, their work also overcomes, use of flooding for finding resources using node management table on each node (similar to directory service on InfA). However, the node management table is constructed by calculating logical similarity of keywords on peers based on primitive form of Jaccard similarity function as opposed to using latent semantics of keywords and finding cosine similarity in our case. Inspired from Crespo *et al.* (Crespo & Garcia-Molina (2004)) Kambayashi *et al.* (Kambayashi & Harada (2009)) also used the terms to capture the realm of resources shared. However, as mentioned before, matching only terms to cannot capture resource characteristics, which is where the author introduced the idea of using latent semantic analysis. In proposed case, the author has introduced the use of minimum support for peer discovery and latent semantic indexing between peers to direct the RA towards resource.

Inspiring from Dasgupta's (Dasgupta (2003, 2008)) and Babaoglu *et al.* (Babaoglu et al. (2002); Babaoglu & Jelasity (2008)) work, Kambayashi *et al.* (Kambayashi & Harada (2009)) introduced the use of pheromone value (AntHill) (that is calculated taking parameters such as number of resources shared by peer and clustering value (logical distance between peers)). This feature is expected to guide search agents towards nodes with high correlation by reducing free-riders. The author believes both, the techniques are equally credible, however the work from Kambayashi *et al.* is discriminating free-rider which may hold a resource that is relevant Kambayashi & Harada (2009). The aim in this work has been to create harmony between nodes and relevance of resources to user's query. The author believes that if resource is available it should be locatable. Finally, they also used DHT - Chord structured P2P system for resource discovery, which the author believes is interesting but opposes the original aim that DHTs cannot handle queries that are multi keyword or text based and is also only viable when keyword for finding resource is known exactly.

Kambayashi *et al.* techniques i.e. guiding search agents using pheromone values and DHT for resource discovery may be leaving "ill-effect" Kambayashi & Harada (2009). In former case, the credible peer by removing free-riders from list of peers that may be holding a resource and in latter case to direct the search agent towards exactly known resource keyword. They are undermining the level of ambiguity and introducing too much certainty into searches which is not the case in proposed system, where user can increase or decrease the search ambiguity/certainty by changing value of *minimum support* thus providing bigger/smaller "canopy" for movement of RAs.

One of the more recent works has been presented by Tan & Zheng (2009). This work offers resource discovery solution, but has no indication of using semantic links for routing the mobile agent. The solution offered seems to be in its earlier stages, implying that all characteristics required by resource discovery system are not answered yet. Though, from this early work it is indicated that there solution also seems to be implemented using FIPA standards.

Other classical work from Dimakopoulos *et al.* has indicated the use of mobile agent as architecture for resource discovery but there synchronisation for distributing local directory (information about shared) resources is done using classical method of flooding, that clearly implies bottleneck of bandwidth limitations and hence saturation of network Dimakopoulos & Pitoura (2003); Chawathe et al. (2003).

5.2 Applications for Research Conducted

Following are examples of few applications that can be developed as a result of this research:

1. Organisation of Documents with Reviewers: Hundreds of documents are submitted to publishers for conference or journal publications that need to be reviewed by the reviewers for finding the worthiness of those documents for that specific conference or journal. The task of matching the documents with reviewers based on their research skills is time consuming and tedious. The outcome of this work can be used by the reviewers to setup their profiles and submit them to the publishers. Upon receiving of documents, the publishers can match, create the keyword list based on content of the document which, when submitted as query will find the appropriate reviewer. So, instead of node-keyword matrix in this application reviewer-keyword matrix will be calculated. This system will perform efficiently and at the same time will offer high degree of effectiveness in terms of find appropriate reviewers.
2. Content-Similarity Check: The purpose of such system ranges from targeted e-marketing to creating clustered documents to comparing two or more documents for similarity. In an e-marketing system e.g. the content of email being received by user can be matched against target advertisements that are of similar domain as the content of the email. In clustered documents, documents belonging to same concept/domain can be organised and furthermore checked for similarity among each other. As the core of

this system is based on LSI-SVD on an overlay network, these services can be extended to large number of nodes.

3. Searching and Locating of Resources: It is not always possible to locate a hosted resource using indexing techniques such as, used by Google. It takes time for web crawlers to scan the newly published website and rank it, resulting into null response if such resource cannot be located. On an overlay network, such as one that is powered by mobile agents, the query initiator need not filter the results obtained to find the suitable resource, once the criteria such as *minimum support* has been provided and that the resource provider is participating on an overlay network, mobile agents can locate a resource dynamically without requirement of web crawlers etc.

The author is sure that there can be many other applications where this system can be applied and implemented. The final product is only limited by a conceivable idea.

5.3 Conclusions

The main objective was to design and implement a resource discovery system that uses mobile agent technology for discovering and selecting nodes and for routing the mobile agent through overlay network based on content of query with purpose of minimising response time, reducing possible delays, maximising network performance by reducing the possibility of saturation and maximising the recall by providing relevant results. Through the conducted research work and the evaluations of experiments in Chapter 4, the author concludes that the process of resource discovery can be improved for P2P system in terms of search performance by increase of recall and hence success rate to resolve queries through use of efficient indexing technique viz. LSI and also that the routing of mobile agent to resolve query through overlay network when supported by heuristics viz. offered using clustering technique will reduce saturation due to higher number of inter-cluster links and decrease response time. The author believes that this resolves the original research question mentioned in Chapter 1.

To summarise the author has proposed a novel resource discovery system that uses mobile agent (RA) for discovering resources on an overlay network that is realised based on semantic similarity of keywords that are shared by peers. The author further proposed a flexible multi-agent based approach to P2P network organisation that is based on the similarity of content shared by peers. The author claims that the use of semantic similarity

between content shared by peers i.e. clustering effect can be effective technique to route the RA to peers that host content that is similar to a user query and finally, that LSI based resource search by RA to find resources hosted by peers that are best match for a user query (where the user query can be text based or an approximate query) is very effective whether the query contains text, that is certain or ambiguous.

The author further demonstrated that proposed approach for resource discovery is better than flooding and further more that an informed search technique used to guide RAs on an overlay network is better than controlled flooding. The results have demonstrated that the using flooding increases the quantity of messages on a network and it can be reduced by use of proposed technique.

In previous experiments, the author used flooding technique to find resource on the network i.e. the RA migrated from one peer to another in hope of finding the resource Singh et al. (2009). The author has realised the shortcoming of last technique and introduced the use of guidance directory on each peer for providing the RA with better chance of finding a resource.

The author realises that initially as resources are scarce, some clusters may not overlap, resulting into cases where resource cannot be located, but the author does understand that as the peers become more resourceful, the clusters will start overlapping to higher degree, hence resulting into better search results.

5.4 Future Work

Although, in ideal case the RA can migrate to suitable nodes and query them for resource, the aspect of breach of security has not been researched in this project. Agents are open to security lapses and hence can be compromised about what to search or what to deliver as result back to query originator. This can jeopardise the integrity of results as well as the RA. Furthermore, the compromised InfA where agent queries about routing for next node for migration can guide the RA migrate towards nodes that do not hold any relevant results. This is an area of future research work that requires attention.

As mentioned before in Section 3.2.1, *keyword-peer* and *keyword-resource* matrices can be large sparse matrices. Holding these large matrices consumes memory which is not always abundant on systems that are continuously publishing or are dynamic. Dimensionality reduction used in proposed work offers a solution to some extent i.e. reduction in matrix

size of an order of around 40%, but that can still be a large matrix. Some research works have been done in this field but are out of scope for this work. Further work can be done in this project to accommodate for this characteristic. The author believes that system architecture presented is very generic and can be further refined in order to support distributed LSI where by the indexing could be decentralised and global search can be conducted for relevant resources on pure P2P overlay network. The problem to generate globally-consistent LSI structure is very challenging as the number of nodes presented by their content is large, dynamic and distributed.

Some research work has been done where local cache is maintained by node to guide the visiting mobile agent so as the computational load for calculating node for migration can be bypassed. It is an interesting feature and can indirectly find its roots in Anthill system used by Babaoglu et al. (2002); Dasgupta (2003); Babaoglu & Jelasity (2008); Kambayashi & Harada (2009). But cache is not always up-to-date and hence can lead to incorrect decisions for migration of mobile agent. In case, the cache can synchronised periodically, this feature can be potentially useful. However, it must be noted that synchronised cache may lead to flooding that increases number of message on network. This area can be studied further to find out its cost-to-benefit ratio.

The author believes that conducted research has far greater potential and can still form foundation for future research work.

References

- Aberer, K., Puceva, M., Hauswirth, M., & Schmidt, R. (2004). Peer-to-peer systems. In *In practical handbook of internet computing*. CRC press.
- Androutsellis-Theotokis, S., & Spinellis, D. (2004, December). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4), 335–371. Available from <http://www.spinellis.gr/pubs/jrnl/2004-ACMCS-p2p/html/AS04.html>
- Arabshian, K., Dickmann, C., & Schulzrinne, H. (2009). The semantic web: Research and applications. In (p. 684-696). Springer Berlin / Heidelberg.
- Babaoglu, O., & Jelasity, M. (2008). Self-* properties through gossiping. *Philosophical Transactions of the Royal Society A*, 366, 3747-3757.
- Babaoglu, O., Meling, H., & Montresor, A. (2002). Anthill: A framework for the development of agent-based peer-to-peer systems. In *Ieee proceedings of 22nd international conference on distributed computing systems (icdcs'02)* (pp. 15–22).
- Bandini, S., Manzoni, S., & Vizzari, G. (2009). Agent based modeling and simulation: An informatics perspective. *Journal of Artificial Societies and Social Simulation*, 12(4), 4. Available from <http://jasss.soc.surrey.ac.uk/12/4/4.html>
- Bawa, M., Manku, G. S., & Raghavan, P. (2003). Sets: search enhanced by topic segmentation. In *Sigir* (p. 306-313).
- Bellifemine, F. L., Caire, G., & Greenwood, D. (2007). *Developing multi-agent systems with jade*. Wiley.
- Bo Chen, J. P., Harry Cheng. (2006). Mobile-c: a mobile agent platform for mobile c/c++ agents. *Software: Practice and Experience*, 36, 1711-1733.

- Casey, J., & Zhou, W. (2009). Reducing the bandwidth requirements of p2p keyword indexing. *International Journal of High Performance Computing and Networking*, 6, 119-129.
- Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., & Shenker, S. (2003). *Making gnutella-like p2p systems scalable*.
- Chen, H., Jin, H., Liu, Y., & Ni, L. M. (2008). Difficulty-aware hybrid search in peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems*, 20, 71-82.
- Crespo, A., & Garcia-Molina, H. (2002). Routing indices for peer-to-peer systems. *Distributed Computing Systems, International Conference on*, 0, 23.
- Crespo, A., & Garcia-Molina, H. (2004). Semantic overlay networks for p2p systems. In *Ap2pc* (p. 1-13).
- Dasgupta, P. (2003). Improving peer-to-peer resource discovery using mobile agent based referrals. In *Ap2pc* (p. 186-197).
- Dasgupta, P. (2008). A multiagent swarming system for distributed automatic target recognition using unmanned aerial vehicles. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 38(3), 549-563.
- Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41, 391-407.
- Dimakopoulos, V. V., & Pitoura, E. (2003). A peer-to-peer approach to resource discovery in multi-agent systems. In *Cooperative information agents* (p. 62-77).
- Doval, D., & O'Mahony, D. (2003). Overlay networks: A scalable alternative for p2p. *IEEE Internet Computing*, 7(4), 79-82.
- Dunne, C. R. (2001). Using mobile agents for network resource discovery in peer-to-peer networks. *ACM SIGecom Exchanges*, 2, 1-9.
- Forum, G. D. (2002). *Gnutella protocol specification v0.4*. Available from <http://rfc-gnutella.sourceforge.net/developer/stable/index.html>

- Gao, J., & Zhang, J. (2005). Clustered svd strategies in latent semantic indexing. *Information Processing and Management*, 41 (5), 1051–1063.
- Golub, G. H., & Loan, C. F. V. (1996). *Matrix computations*. The John Hopkins University Press.
- Guvnec, I., & Urdaneta, J. J. (2010). *Peer-to-peer file sharing: A survey*. Available from <http://www.cs.ucr.edu/~michalis/COURSES/179-03/p2psurvey.ppt>
- Hasan, M., & Matsumoto, Y. (1999). *Document clustering: before and after the singular value decomposition* (Tech. Rep.). Nara Institute of Science and Technology. Technical Report TR-99. Processing Society of Japan.
- Kambayashi, Y., & Harada, Y. (2007). A resource discovery method based on multi-agents in p2p systems. In *Kes-amsta* (p. 364-374).
- Kambayashi, Y., & Harada, Y. (2009). A resource discovery method based on multiple mobile agents in p2p systems. In *Intelligent agents in the evolution of web and applications* (p. 113-135).
- Kang, S., Lee, Y., Lee, D., & Youn, H. Y. (2007). A landmark-based scalable semantic resource discovery scheme. *IEICE - Trans. Inf. Syst.*, E90-D(6), 986–989.
- Karnstedt, M., Hose, K., & Sattler, K. uwe. (2004). Query routing and processing in schema-based p2p systems. In *In proceedings of dexa workshops* (pp. 544–548). IEEE Computer Society.
- Killmeyer, J. (2006). *Information security architecture : an integrated approach to security in the organization* (2nd ed. ed.). Auerbach. Boca Raton, Fla. u.a.
- Konchady, M. (2006). *Text mining application programming*. Charles River Media.
- Li, J., Loo, B. T., Hellerstein, J. M., Kaashoek, M. F., Karger, D. R., & Morris, R. (2003). On the feasibility of peer-to-peer web indexing and search. In *Iptps* (p. 207-215).
- Lingnau, A., & Drobnik, O. (1999). Simulating mobile agent systems with swarm. *Agent Systems and Applications, International Symposium on / International Symposium on Mobile Agents*, 0, 272.

- Liu, X., Chen, M., & Yang, G. (2004). Latent semantic indexing in peer-to-peer networks. In *Arcs* (p. 63-77).
- Lopes, A. L., & Botelho, L. M. (2008). Improving multi-agent based resource coordination in peer-to-peer networks. *Journal of Networks*, 3(2), 38-47.
- Ly, Q., Cao, P., Cohen, E., Li, K., & Shenker, S. (2002). Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th annual acm international conference on supercomputing (ics) 2002*.
- Mastroianni, C., Talia, D., & Verta, O. (2005). A super-peer model for building resource discovery services in grids: Design and simulation analysis. In *Egc* (p. 132-143).
- McCrary, K., & Waters, B. (2000, October). *Jtella v0.7*. Available from <http://jtella.sourceforge.net/>
- Milojicic, D. S., Breugst, M., Busse, I., Campbell, J., Covaci, S., Friedman, B., et al. (1998). Masif: The omg mobile agent system interoperability facility. *Personal and Ubiquitous Computing*, 2(2).
- Napster. (2003). *Napster*. Available from <http://www.napster.com>
- North, M. J., Collier, N. T., & Vos, J. R. (2006). Experiences creating three implementations of the repast agent modeling toolkit. *ACM Trans. Model. Comput. Simul.*, 16(1), 1-25.
- Prakash, A. (2006, May). *A survey of advanced search in p2p networks*. Department of Computer Science. Available from <http://www.medianet.kent.edu/surveys/IAD06S-p2psearch-alok/index.html>
- Project, T. C. (2010). *Chord faq*. Available from <http://pdos.csail.mit.edu/>
- Ratnasamy, S., Francis, P., Shenker, S., Karp, R., & Handley, M. (2001). A scalable content-addressable network. In *In proceedings of acm sigcomm* (pp. 161-172).
- Reynolds, P., & Vahdat, A. (2003). Efficient peer-to-peer keyword searching. In *Middleware '03: Proceedings of the acm/ifip/usenix 2003 international conference on middleware* (pp. 21-40). New York, NY, USA: Springer-Verlag New York, Inc.
- Rowstron, A., & Druschel, P. (2001). *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*.

- Schoeman, M., & Cloete, E. (2003). Architectural components for the efficient design of mobile agent systems. In *Saicsit '03: Proceedings of the 2003 annual research conference of the south african institute of computer scientists and information technologists on enablement through technology* (pp. 48–58). , Republic of South Africa: South African Institute for Computer Scientists and Information Technologists.
- Singh, M., Cheng, X., & He, X. (2009). *Multimedia resource discovery using mobile agent*. New York,: IGI Global.
- Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., et al. (2001). Chord: A scalable peer-to-peer lookup protocol for internet applications. In *Acm sigcomm* (pp. 149–160).
- Sun, Y., Sun, L., Huang, X., & Lin, Y. (2006). Resource discovery in locality-aware group-based semantic overlay of peer-to-peer networks. In *Infoscale '06: Proceedings of the 1st international conference on scalable information systems* (p. 40). New York, NY, USA: ACM.
- Talai, P. T. D., Fragopoulou, P., Mordacchini, M., Pennanen, M., Popov, K., & Haridi, V. V. S. (2006). *Peer-to-peer models for resource discovery on grids* (Tech. Rep.). Institution of System Architecture.
- Tan, Y., & Zheng, Z. (2009). A multi-agent based resource discovery scheme for p2p systems. In *International workshop on intelligent systems and applications - isa 2009* (p. 1-4).
- Tang, C., & Dwarkadas, S. (2004). Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Nsdi* (p. 211-224).
- Tang, C., Xu, Z., & Dwarkadas, S. (2003). Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceeding of acm sig-comm* (p. 178-186).
- Tran, H. M., & Schonwalden, J. (2008, March). *Distributed case-based reasoning for fault management*. 1st EMANICS Workshop on Peer-to-Peer Management.
- Trillo, R., Ilarri, S., & Mena, E. (2007). Comparison and performance evaluation of mobile agent platforms. In *Icas '07: Proceedings of the third international conference on autonomic and autonomous systems* (p. 41). Washington, DC, USA: IEEE Computer Society.

- University, C. (1999, August). *Cornell university: Medline text collection*. Smart System. USA. Available from <ftp://ftp.cs.cornell.edu/pub/smart/med/>
- Vieira, R. (2001). *Foundation of intelligent agents - agent communication language* (Tech. Rep.). FIPA.
- Wireshark. (2010). *Wireshark go deep*. Online. Available from <http://www.wireshark.org>
- Wong, J., Helmer, G., Naganathan, V., Polavarapu, S., Honavar, V. G., & Miller, L. (2001). Smart mobile agent facility. *Journal of Systems and Software*, 56(1), 9–22.
- Yang, K.-H., Wu, C.-J., & Ho, J.-M. (2007). Antsearch: An ant search algorithm in unstructured peer-to-peer networks. *IEICE Transactions*, 89-B(9), 2300–2308.
- Yingwu Zhu, Y. H. (2005). *Handbook on theoretical and algorithmic aspects of sensor, ad hoc wireless, and peer-to-peer networks - semantic search in peer-to-peer systems* (J. Wu, Ed.). CRC Press.
- Zaharia, M., & Keshav, S. (2008). Gossip-based search selection in hybrid peer-to-peer networks: Research articles. *Concurr. Comput. : Pract. Exper.*, 20(2), 139–153.
- Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., & Kubiawicz, J. D. (2004). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22, 41–53.
- Zhao, B. Y., Kubiawicz, J., Joseph, A. D., Zhao, B. Y., Kubiawicz, J., & Joseph, A. D. (2001). *Tapestry: An infrastructure for fault-tolerant wide-area location and routing* (Tech. Rep.).
- Zhong, Y., & Liu, J. (2003). *The mobile agent technology*. ISBN 7-89494-143-3.
- Zhu, Y., & Hu, Y. (2004). *Ess: Efficient semantic search on gnutella-like p2p systems* (Tech. Rep.). Department of ECECS, University of Cincinnati.
- Zhu, Y., & Hu, Y. (2005). Efficient, proximity-aware load balancing for dht-based p2p systems. *IEEE Trans. Parallel Distrib. Syst.*, 16(4), 349–361.
- Zhu, Y., & Hu, Y. (2006). Handbook of theoretical and algorithmic aspects of ad hoc, sensor, and peer-to-peer networks. In J. Wu (Ed.), (p. 634–664). Auerbach Publications.

- Zhu, Y., & Hu, Y. (2007). Efficient semantic search on dht overlays. *Journal of Parallel and Distributed Computing*, 67(5), 604 - 616.
- Zhu, Y., Wang, H., & Hu, Y. (2003). Intergrating semantics-based access mechanisms with p2p file systems. In *Proceedings of third international conference on peer-to-peer computing*.

APPENDIX A

SIMILARITY MEASURES AND WEIGHTING FUNCTIONS

Assuming two n -dimensional vectors $X = (x_1, x_2, x_3, \dots, x_n)$ and $Y = (y_1, y_2, y_3, \dots, y_n)$.

Name of Measure	Formula
Euclidean Distance	$\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$
Dot Product	$\sum_{i=1}^n x_i y_i$
Jaccard Similarity	$\frac{ X \cap Y }{ X \cup Y }$
Cosine Similarity	$\frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i} \sqrt{\sum_{i=1}^n y_i}}$

TABLE A.1: Similarity measures

Local Weighting Functions $L(m, n)$ and Global Weighing Functions $G(m)$:

Type	$L(m, n)$	$G(m)$
Binary	$\begin{cases} 0 & tf_{mn} = 0 \\ 1 & tf_{mn} > 0 \end{cases}$	$\sqrt{\frac{1}{\sum_n (tf_{mn})^2}}$
Term-Frequency	tf_{mn}	$\frac{GlobalFrequencyOfTerm"m"}{FrequencyOfNodesInWhichTerm"m"Appears}$
log	$\ln(tf_{mn} + 1)$	$\ln(\frac{NumberOfDocuments}{FrequencyOfNodesInWhichTerm"m"Appears}) + 1$

TABLE A.2: Local and global weighting functions

Where

tf_{mn} = Frequency of term m in node n

APPENDIX B

CLASSES REALISED - AFFINITY

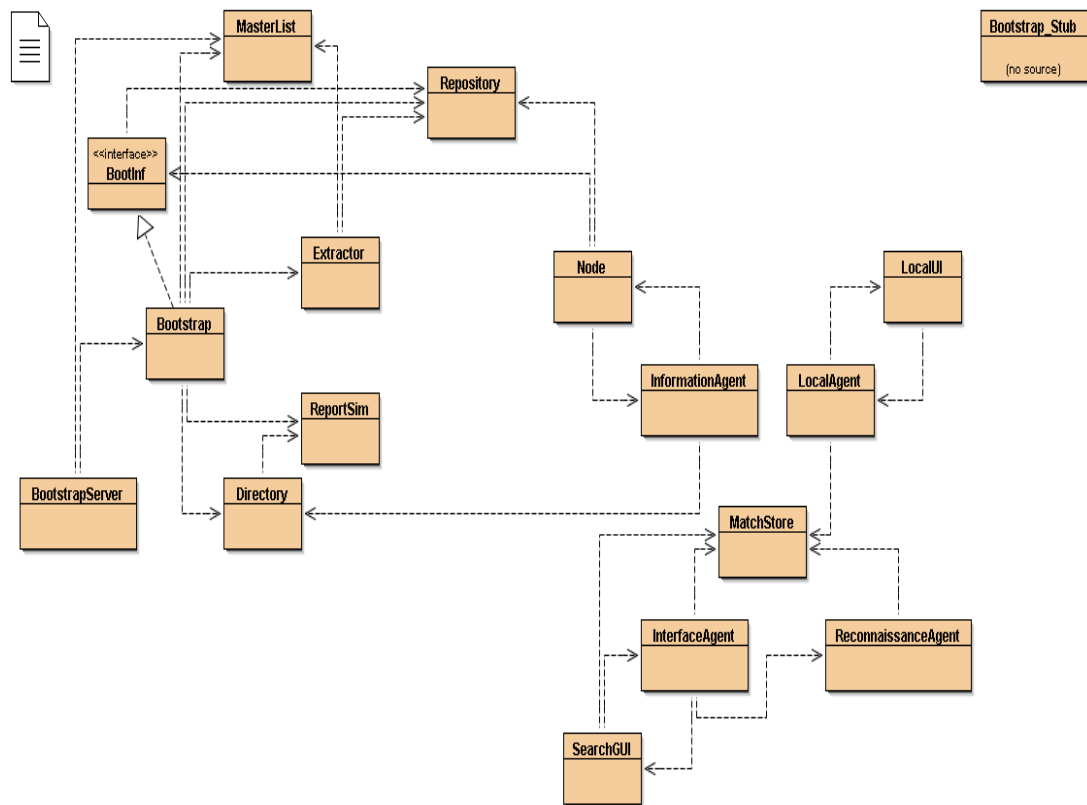


FIGURE B.1: Classes for resource discovery system - Affinity

APPENDIX C

PROGRAM LISTINGS - AFFINITY

C.1 Interface BootInf.java

```
1  import java.rmi.*;
2  import java.util.*;
3  /**
4   * Remote RMI Interface for Bootstrap Server
5   *
6   * @author M. Singh
7   * @version 1.0
8   */
9
10 public interface BootInf extends Remote
11 {
12     public HashMap<String,Directory> register(Repository message, double
13         min_Sup) throws RemoteException;
14     public void disconnect(String registrationIP) throws RemoteException;
15 }
```

C.2 Class Bootstrap.java

```

1  import java.rmi.*;
2  import java.rmi.server.*;
3  import java.util.*;
4  import Jama.*;
5  /**
6   * Implementation of BootInf Remote Methods. These methods available to
7   * Information Agent for registering the Node
8   *
9   * @author M. Singh
10  * @version 2.0
11  */
12 public class Bootstrap extends UnicastRemoteObject implements BootInf
13 {
14     private MasterList database;
15     private Extractor extractor;
16     private HashMap<String, Directory> map;
17
18     public Bootstrap(MasterList database) throws RemoteException
19     {
20         this.database=database;
21         /**
22          * Dummy Repository added to compensate for null pointer
23          */
24         String dummyIP="bootstrap";
25         ArrayList dummyKeyword = new ArrayList();
26         dummyKeyword.add("ytiniffa");
27         dummyKeyword.add("metsys");
28         //dummyKeyword.add("shipment");
29         //dummyKeyword.add("of");
30         //dummyKeyword.add("gold");
31         //dummyKeyword.add("damaged");
32         //dummyKeyword.add("in");
33         //dummyKeyword.add("a");
34         //dummyKeyword.add("fire");
35         Repository dummyMessage = new Repository(dummyIP, dummyKeyword);
36         database.add(dummyIP);
37         extractor = new Extractor(database, dummyMessage);
38     }

```

```

39
40     public HashMap<String,Directory> register(Repository message, double
        min_Sup) throws RemoteException
41     {
42         System.out.println( "\nSubmitted_keywords_by_client_"+message.getIP()+"
            \n"+message.getKeywords());
43         //Add ip to MasterList
44         database.add(message.getIP());
45         extractor = new Extractor(database, message);
46         //Prepare Reply for the Client based on its preferences
47         HashMap<String,Directory> hashmap = prepareSimReply(min_Sup);
48         return hashmap;
49     }
50
51     public HashMap<String,Directory> prepareSimReply(double min_Sup)
52     {
53         map=new HashMap<String,Directory>();
54         //ReportSim report = new ReportSim();
55         Matrix sim = database.getSimMatrix();
56         ArrayList clients = database.getClientList();
57         Matrix master = database.getMasterKeywordList();
58
59         ArrayList indexHolder = new ArrayList();
60
61         //find index in sim report that is higher than user minimum support
            preference
62         double [][] simArray = sim.getArray();
63         for (int a=0;a<sim.getRowDimension();a++)
64         {
65             if (simArray[a][0]>min_Sup)
66             {
67                 indexHolder.add(a);
68             }
69         }
70         ArrayList<Directory> directoryHolder=new ArrayList<Directory>();
71         for (int i=0;i<indexHolder.size();i++)
72         {
73             Directory directory = new Directory();
74             directoryHolder.add(directory);
75         }

```

```

76      //set similarity sub matrix based on index holder
77      Object [] rowI = indexHolder.toArray();
78      int [] rows = new int [rowI.length];
79      int [] cols = {0};
80      for (int i=0;i<rows.length;i++)
81      {
82          Integer r = (Integer)rowI[i];
83          int rs = r.intValue();
84          rows[i]=rs;
85      }
86      Matrix tempSim = sim.getMatrix(rows, cols);
87      System.out.println("SIMILARITY_MATRIX_FOR_LATEST_CLIENT");
88      tempSim.print(tempSim.getColumnDimension(),3);
89      //report.setSimilarity(tempSim);
90      double [][] arraySim = tempSim.getArray();
91      for (int i=0;i<directoryHolder.size();i++)
92      {
93          Directory d = directoryHolder.get(i);
94          d.similarityValue=arraySim[i][0];
95          directoryHolder.set(i,d);
96      }
97
98      //set clients based on index holder
99      ArrayList tempClients = new ArrayList();
100     for (int i=0;i<indexHolder.size();i++)
101     {
102         tempClients.add(((String)clients.get(((Integer)indexHolder.get(i)).
            intValue())));
103     }
104     System.out.println("CLIENTS_WITH_BEST_SIMILARITY_-_IN_CLUSTER");
105     System.out.println(tempClients);
106     //report.setClients(tempClients);
107
108     //set temp master sub matrix based on index holder
109     Object [] colI = indexHolder.toArray();
110     int [] colm = new int [colI.length];
111     for (int i=0;i<colm.length;i++)
112     {
113         Integer c = (Integer)colI[i];
114         int cs= c.intValue();

```

```

115         colm[i]=cs;
116     }
117     int [] rowm = new int [ master.getRowDimension() ];
118     for ( int i=0;i<rowm.length;i++)
119     {
120         rowm[i]=i;
121     }
122     Matrix tempWeights = master.getMatrix(rowm,colm);
123     System.out.println( "WEIGHTS_FOR_CLIENTS_KEYWORDS_-_IN_CLUSTER" );
124     tempWeights.print( tempWeights.getColumnDimension(),1 );
125     //report.setKeywordsWeights( tempWeights );
126     double [][] tempWeightsArray = tempWeights.getArray();
127     for ( int m=0;m<tempWeightsArray[0].length;m++)
128     {
129         Directory d = directoryHolder.get(m);
130         double [][] keywordWeight = new double [tempWeightsArray.length][1];
131         for ( int n=0;n<tempWeightsArray.length;n++)
132         {
133             keywordWeight[n][0]=tempWeightsArray[n][m];
134         }
135         Matrix keyWeight = new Matrix(keywordWeight);
136         d.keyWeights= keyWeight;
137         directoryHolder.set(m,d);
138     }
139
140     //set keyword List
141     //report.setKeywordList( database.getKeywordList() );
142     for ( int i=0;i<directoryHolder.size();i++)
143     {
144         Directory d = directoryHolder.get(i);
145         d.keywords=database.getKeywordList();
146         directoryHolder.set(i,d);
147     }
148
149     /**
150     * Test Purpose Only
151     */
152     /*
153     for ( int h=0;h<directoryHolder.size();h++)
154     {

```

```
155         Directory b = directoryHolder.get(h);
156         System.out.println("SEEMS UPDATED");
157         System.out.println(b.similarityValue);
158         Matrix a = b.keyWeights;
159         a.print(a.getColumnDimension(),1);
160         System.out.println(b.keywords);
161     }
162     */
163
164     for (int i=0;i<directoryHolder.size();i++)
165     {
166         Directory d = directoryHolder.get(i);
167         //d.report = report;
168         directoryHolder.set(i,d);
169     }
170
171     for (int i=0;i<tempClients.size();i++)
172     {
173         map.put((String)tempClients.get(i),directoryHolder.get(i));
174     }
175
176     return map;
177 }
178
179 public void disconnect(String registrationIP) throws RemoteException
180 {
181     System.out.println("Removed_IP_"+registrationIP);
182     map.remove(registrationIP);
183     database.remove(registrationIP);
184 }
185
186 }//end class
```


C.3 Class BootstrapServer.java

```

1  import java.rmi.*;
2  import java.util.*;
3  import java.net.*;
4  /**
5   * RMI Bootstrap Server
6   *
7   * @author M. Singh
8   * @version 1.1
9   */
10 public class BootstrapServer
11 {
12     public static void main(String argv[])
13     {
14         String localIP="";
15         String reference="";
16         try
17         {
18             InetAddress local_Address = InetAddress.getLocalHost();
19             localIP = local_Address.getHostAddress();
20         } catch (java.net.UnknownHostException e)
21         {
22             System.out.println("Error_getting_IP_Address_"+e);
23         }
24
25         try
26         {
27             //ArrayList<Repository> database = new ArrayList<Repository>();
28             MasterList database = new MasterList();
29             Bootstrap bootstrap = new Bootstrap(database);
30             reference = "rmi://" + localIP + "/Server_1";
31             Naming.rebind(reference, bootstrap);
32             System.out.println("Bootstrap_server_instance_"+reference+"
33                               Running_\nWaiting_for_Nodes_to_regster");
34         } catch (Exception e)
35         {
36             System.out.println("Error_Starting_Bootstrap_Server_"+e);
37         }
38     }

```

```
38  }//end class
```

C.4 Class Extractor.java

```

1  import java.util.*;
2  /**
3   * Extractor is helper class for MasterList used for extracting keywords from
      Repository
4   *
5   * @author M. Singh
6   * @version 1.5
7   */
8  public class Extractor
9  {
10     private MasterList database;
11     private Repository message;
12     private String tk[];
13     private int size;
14     private SortedMap map;
15
16     public Extractor(MasterList database, Repository message)
17     {
18         this.database=database;
19         this.message=message;
20         map=new TreeMap();
21         getKeywords();
22     }
23
24     public void getKeywords()
25     {
26         int index=0;
27         String keywordSet=" ";
28         ArrayList keywords = message.getKeywords();
29         for (int i=0;i<keywords.size();i++)
30         {
31             keywordSet+=(String)keywords.get(i)+"_";
32         }
33         //Tokenize
34         StringTokenizer token = new StringTokenizer(keywordSet);
35         size=token.countTokens();
36         makeTKArray(size);
37         while(token.hasMoreTokens())

```

```
38         {
39             tk[index]=token.nextToken();
40             findTokenFrequency(tk[index]);
41         }
42         database.addKeywords(map);
43     }
44
45     public void makeTKArray(int size)
46     {
47         tk = new String[size];
48     }
49
50     public void findTokenFrequency(String token)
51     {
52         if(!map.containsKey(token))
53         {
54             map.put(token.toLowerCase(),1);
55         } else
56         {
57             Integer frequency = (Integer)map.get(token);
58             int freqVal = frequency.intValue();
59             freqVal+=1;
60             map.remove(token);
61             map.put(token, freqVal);
62         }
63     }
64
65 }
```

C.5 Class ReportSim.java

```
1  import Jama.*;
2  import java.util.*;
3  import java.io.*;
4  /**
5   * Serialized class Similarity Report sent between Bootstrap Server and Node
6   *
7   * @author M. Singh
8   * @version 1.1
9   */
10 public class ReportSim implements Serializable
11 {
12     private ArrayList clients;
13     private Matrix similarity;
14     private Matrix weights;
15     private ArrayList keywordList;
16
17     public ReportSim(ArrayList clients, Matrix similarity, Matrix weights,
18                     ArrayList keywordList)
19     {
20         this.clients=clients;
21         this.similarity=similarity;
22         this.weights=weights;
23         this.keywordList=keywordList;
24     }
25
26     public ReportSim()
27     {
28
29     }
30
31     public void setClients(ArrayList clients)
32     {
33         this.clients=clients;
34     }
35
36     public ArrayList getClients()
37     {
38         return clients;
39     }
39 }
```

```
38
39     public void setSimilarity(Matrix similarity)
40     {
41         this.similarity=similarity;
42     }
43
44     public Matrix getSimilarity()
45     {
46         return similarity;
47     }
48
49     public void setKeywordsWeights(Matrix weights)
50     {
51         this.weights=weights;
52     }
53
54     public Matrix getKeywordsWeights()
55     {
56         return weights;
57     }
58
59     public void setKeywordList(ArrayList keywordList)
60     {
61         this.keywordList=keywordList;
62     }
63
64     public ArrayList getKeywordList()
65     {
66         return keywordList;
67     }
68 }
```

C.6 Class Directory.java

```
1  import java.io.*;
2  import Jama.*;
3  import java.util.*;
4  /**
5   * Data Structure for holding the directory peer – keyword matrix used by
6   * Information Agent and Bootstrap
7   *
8   * @author M. Singh
9   * @version 1.1
10  */
11 public class Directory implements Serializable
12 {
13     double similarityValue;
14     Matrix keyWeights;
15     ArrayList keywords;
16     //ReportSim report;
17 }
```

C.7 Class MasterList.java

```

1  import java.util.*;
2  import Jama.*;
3  /**
4   * Master List class holds the global peer - keyword matrix
5   *
6   * @author M. Singh
7   * @version 2.1
8   */
9  public class MasterList
10 {
11     private ArrayList ipCols;
12     private ArrayList<SortedMap> tempMaster;
13     private double [][] masterKeywordMatrix;
14     private double [][] joiningNodeKeywords;
15     private ArrayList<String> listOfKeywords;
16     private Matrix U;
17     private Matrix S;
18     private Matrix V;
19     private Matrix S_inverse;
20     private Matrix V_transpose;
21     private Matrix Q_transpose;
22     private Matrix q;
23     private Matrix simM;
24     private String forIP=" ";
25
26     public MasterList ()
27     {
28         ipCols = new ArrayList ();
29         tempMaster = new ArrayList<SortedMap>();
30     }
31
32     public void add(String ip)
33     {
34         //ipCols.add(ip);
35         if (!ipCols.contains(ip))
36         {
37             ipCols.add(ip);
38             forIP=ip;

```



```

39         } else
40         {
41             int index=ipCols.indexOf(ip);
42             ipCols.remove(index);
43             tempMaster.remove(index);
44             ipCols.add(ip);
45             forIP=ip;
46         }
47     }
48
49     public void remove(String ip)
50     {
51         if(ipCols.contains(ip))
52         {
53             int index=ipCols.indexOf(ip);
54             ipCols.remove(index);
55             tempMaster.remove(index);
56         }
57     }
58
59     public void addKeywords(SortedMap map)
60     {
61         tempMaster.add(map);
62         prepareMatrix();
63     }
64
65     public void prepareMatrix()
66     {
67         SortedMap completeList = new TreeMap();
68         SortedMap temp = new TreeMap();
69         for(int i=0;i<tempMaster.size();i++)
70         {
71             temp=tempMaster.get(i);
72             Set keywords = temp.keySet();
73             Iterator itKeys = keywords.iterator();
74             while(itKeys.hasNext())
75             {
76                 String key = (String)itKeys.next();
77                 if(!completeList.containsKey(key))
78                 {

```

```

79         completeList.put(key,0);
80     }
81     }//end while
82 }//end loop
83
84 setCompleteList( completeList );
85
86 masterKeywordMatrix = new double[ completeList.size() ][ ipCols.size() ];
87 int rows = masterKeywordMatrix.length;
88 int cols = masterKeywordMatrix[0].length;
89 for( int n=0;n<cols;n++)
90 {
91     temp= new TreeMap();
92     Set completeKeys = completeList.keySet();
93     Iterator it = completeKeys.iterator();
94     temp=tempMaster.get(n);
95     for( int m=0;m<rows;m++)
96     {
97         while( it.hasNext())
98         {
99             String key = (String)it.next();
100             if( temp.containsKey( key ))
101             {
102                 int val = (( Integer)temp.get( key )).intValue();
103                 masterKeywordMatrix[m][ n]=val;
104             } else
105             {
106                 masterKeywordMatrix[m][ n]=0.0;
107             }
108             m++;
109         }
110     }
111 }
112
113 if( cols==1)
114 {
115     //do nothing
116     //to protect system from issuing dummy
117 } else
118 {

```

```

119      //display
120      System.out.println("Master_Matrix");
121      displayMatrix(masterKeywordMatrix,cols);
122      //calculate SVD considering the joining column ip address and its
           keywords is the new query(joining node keywords).
123      double [][] tempKeywordMatrix = new double[rows][cols-1];
124      joiningNodeKeywords=new double[rows][1];
125      int runner=0;
126      for (int b=0;b<cols;b++)
127      {
128          for (int a=0;a<rows;a++)
129          {
130              if(b==ipCols.indexOf(forIP))
131              {
132                  //do not get that column
133                  //make it joining node
134                  //joined keywords
135                  for (int z=0;z<rows;z++)
136                  {
137                      joiningNodeKeywords[z][0]=masterKeywordMatrix[z][b]
138                          ];
139                  }
140              } else
141              {
142                  tempKeywordMatrix[a][runner]=masterKeywordMatrix[a][b]
143                      ];
144              }
145          }
146          runner++;
147      }
148      /*
149      for (int b=0;b<cols-1;b++)
150      {
151          for (int a=0;a<rows;a++)
152          {
153              tempKeywordMatrix[a][b]=masterKeywordMatrix[a][b];
154          }
155      }

```

```

156         //joined keywords
157         for ( int a=0;a<rows;a++)
158         {
159             joiningNodeKeywords[a][0]=masterKeywordMatrix[a][ ipCols.size()
160                 -1];
161         }
162         */
163         System.out.println("Compared_Against_Matrix");
164         displayMatrix(tempKeywordMatrix,tempKeywordMatrix[0].length);
165         System.out.println("Joining_Matrix");
166         displayMatrix(joiningNodeKeywords,joiningNodeKeywords[0].length);
167         Matrix Q = new Matrix(joiningNodeKeywords);
168         Q_transpose=Q.transpose();
169         calculateSVD(tempKeywordMatrix);
170         calculateq();
171         Matrix sim = calculateSim();
172         System.out.println("Similarity_Report");
173         sim.print(sim.getColumnDimension(),3);
174     }
175 }
176
177 /**
178  * Displays matrix
179  */
180 public void displayMatrix(double[][] matrix,int cols)
181 {
182     Matrix mat = new Matrix(matrix);
183     mat.print(cols,1);
184 }
185
186 /**
187  * SVD calculation
188  */
189 public void calculateSVD(double[][] matrix)
190 {
191     Matrix mat = new Matrix(matrix);
192     SingularValueDecomposition svd = mat.svd();
193     U = svd.getU();
194     //U.print(ipCols.size(),3);

```

```

195         S = svd.getS();
196         //S.print(ipCols.size(),3);
197         S_inverse = S.inverse();
198         V = svd.getV();
199         //V.print(ipCols.size(),3);
200         V_transpose= V.transpose();
201     }
202
203     /**
204      * Computing query vector
205      */
206     public void calculateq()
207     {
208         q = (Q_transpose.times(U)).times(S_inverse);
209     }
210
211     public Matrix calculateSim()
212     {
213         double[][] vofQuery = q.toArray();
214         double[][] vofTerm = V_transpose.toArray();
215         double[][] sim=new double[vofQuery[0].length][1];
216         double[] num=new double[vofTerm.length];
217         double den1=0;
218         double[] den2=new double[vofTerm.length];
219
220         for (int x =0;x<vofTerm.length;x++)
221         {
222             for (int i =0;i<vofQuery[0].length;i++)
223             {
224                 num[x] +=vofQuery[0][i]*vofTerm[i][x];
225             }
226         }
227
228         for (int x=0;x<vofQuery[0].length;x++)
229         {
230             den1+=vofQuery[0][x]*vofQuery[0][x];
231         }
232
233         den1 = Math.sqrt(den1);
234

```

```

235         for ( int i=0; i<den2.length; i++)
236         {
237             for ( int x=0; x<vofTerm.length; x++)
238             {
239                 den2[i] += vofTerm[x][i] * vofTerm[x][i];
240             }
241         }
242
243         for ( int x=0; x<den2.length; x++)
244         {
245             den2[x] = Math.sqrt(den2[x]);
246         }
247
248         for ( int i=0; i<sim.length; i++)
249         {
250             sim[i][0] = num[i] / (den1 * den2[i]);
251         }
252
253         simM = new Matrix(sim);
254         return simM;
255     }
256
257     public Matrix getSimMatrix()
258     {
259         return simM;
260     }
261
262     public ArrayList getClientList()
263     {
264         return ipCols;
265     }
266
267     public Matrix getMasterKeywordList()
268     {
269         Matrix master = new Matrix(masterKeywordMatrix);
270         return master;
271     }
272
273     public void setCompleteList(SortedMap completeList)
274     {

```

```
275         Set keys = completeList.keySet();
276         listOfKeywords = new ArrayList<String>();
277         Iterator it = keys.iterator();
278         while( it.hasNext() )
279         {
280             listOfKeywords.add((String) it.next());
281         }
282     }
283
284     public ArrayList getKeywordList()
285     {
286         return listOfKeywords;
287     }
288 } //end class
```

C.8 Class Repository.java

```
1  import java.io.*;
2  import java.util.*;
3  /**
4   * Serialised Repository Data Structure
5   *
6   * @author M. Singh
7   * @version 1.0
8   */
9  public class Repository implements Serializable
10 {
11     private String ip_Address;
12     private ArrayList keywords;
13
14     public Repository(String ip_Address, ArrayList keywords)
15     {
16         this.ip_Address=ip_Address;;
17         this.keywords=keywords;
18     }
19
20     public void setIP(String ip_Address)
21     {
22         this.ip_Address=ip_Address;
23     }
24
25     public String getIP()
26     {
27         return ip_Address;
28     }
29
30     public void setKeywords(ArrayList keywords)
31     {
32         this.keywords=keywords;
33     }
34
35     public ArrayList getKeywords()
36     {
37         return keywords;
38     }
```


39

40 }//end class

C.9 Class Node.java

```

1  import java.rmi.*;
2  import java.net.*;
3  import java.util.*;
4  import jade.core.*;
5  /**
6   * Node class presents the peer and is responsible for communication with RMI
       Bootstrap server
7   * and register the peer and keyword matrix
8   *
9   * @author M. Singh
10  * @version 1.0
11  */
12 public class Node
13 {
14     String name="";
15     String reference="";
16     BootInf boot=null;
17     InformationAgent ia;
18     HashMap<String,Directory> clusterNeighbours;
19
20     public Node()
21     {
22         try
23         {
24             reference = "rmi://192.168.1.144/Server_1";
25             boot = (BootInf)Naming.lookup(reference);
26             System.out.println("Connected_to_"+reference+"_successfully.");
27         } catch (Exception e)
28         {
29             System.out.println("Error_on_Node_"+e);
30             try
31             {
32                 //boot.disconnect(name);
33             } catch (Exception ee)
34             {
35                 System.out.println(e);
36             }
37         }

```

```
38     }
39
40     public void setAgent(InformationAgent agent)
41     {
42         this.ia=ia;
43     }
44
45     public void connectToBootStrap(String nameLocalAgent, ArrayList<String>
        keywords)
46     {
47         name=nameLocalAgent;
48         try
49         {
50             Repository message = new Repository(nameLocalAgent, keywords);
51             clusterNeighbours = boot.register(message,0.0);
52         } catch (Exception e)
53         {
54             try
55             {
56                 //boot.disconnect(nameLocalAgent);
57             } catch (Exception ee)
58             {
59                 System.out.println("Error_Disconnecting_"+ee);
60             }
61         }
62     }
63
64     public void remove()
65     {
66         try
67         {
68             boot.disconnect(name);
69             System.exit(0);
70         } catch (Exception e)
71         {
72             System.out.println(e);
73         }
74     }
75
76     public HashMap<String, Directory> getNeighbours()
```

```
77     {  
78         return clusterNeighbours;  
79     }  
80 }//end class
```

C.10 Class InformationAgent.java

```

1  import jade.core.*;
2  import jade.lang.acl.*;
3  import jade.core.behaviours.*;
4  import java.util.*;
5  import javax.swing.*;
6  import Jama.*;
7
8  /**
9   * Information Agent holds information about peers that are semantically
10    similar to this peer.
11   * @author M. Singh
12   * @version 1.5
13   */
14 public class InformationAgent extends Agent
15 {
16     private String nameLA="";
17     private ArrayList<String> keywords;
18     private Node node=new Node();
19     private HashMap<String,Directory> clusterNeighbours;
20     private String cont;
21     private String query;
22     private String minSup;
23     private Database database = new Database();
24     private Finder finder;
25
26     protected void setup()
27     {
28         //Display the GUID name of agent
29         String name = getAID().getName();
30         System.out.println("Information-agent_GUID_"+name+"_started.");
31         //there for the IA name must be
32         if(name.startsWith("I"))
33         {
34             nameLA=name;
35             nameLA=nameLA.replace("I","L");
36         }
37         node.setAgent(this);

```

```

38
39         addBehaviour(new KeywordRequestor(this));
40         addBehaviour(new NodeRequestor());
41     }
42
43     protected void takeDown()
44     {
45         doDelete();
46     }
47
48     public void callNodeRegistry()
49     {
50         if (keywords != null)
51         {
52             node.connectToBootStrap(nameLA, keywords);
53         }
54     }
55
56     //inner class Keyword Requestor
57     private class KeywordRequestor extends TickerBehaviour
58     {
59         private KeywordRequestor(Agent a)
60         {
61             super(a, 20000);
62         }
63
64         public void onStart()
65         {
66             // some thing for start
67         }
68
69         public void onTick()
70         {
71             //Send Message
72             ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
73             request.addReceiver(new AID(nameLA, AID.ISGUID));
74             request.setConversationId("keywords-request");
75             request.setReplyWith("request" + System.currentTimeMillis());
76             myAgent.send(request);
77

```

```

78      //Prepare message receiving template
79      MessageTemplate mt = MessageTemplate.and(MessageTemplate.
          MatchConversationId( "keywords-request" ), MessageTemplate.
          MatchInReplyTo( request.getReplyWith() ));
80      ACLMessage reply = myAgent.receive();
81
82      if( reply!= null)
83      {
84          if( reply.getPerformative()==ACLMessage.INFORM)
85          {
86              try
87              {
88                  keywords=(ArrayList<String>)reply.getContentObject();
89              } catch( Exception e)
90              {
91                  e.printStackTrace();
92              }
93          }
94          callNodeRegistry();
95      }
96      else
97      {
98          //System.out.println("This will take 60000 msecs - Current
          //State Block");
99          //block();
100      }
101    }
102  } //end inner class
103
104  //inner class Node Requestor
105  private class NodeRequestor extends CyclicBehaviour
106  {
107      private MessageTemplate mt = MessageTemplate.MatchPerformative(
          ACLMessage.REQUEST);
108      public void action()
109      {
110          clusterNeighbours=node.getNeighbours();
111          try
112          {
113              ACLMessage messagerec=myAgent.receive(mt);

```

```

114         if (messagerec!=null)
115     {
116         System.out.println("Request_from_"+messagerec.getSender().
            getLocalName()+"\n"+"reconnaissance_agent_to_issue_the
            _node");
117         String cont = messagerec.getContent();
118         String [] myCont = cont.split(":");
119         query=myCont[0];
120         minSup=myCont[1];
121         System.out.println("QUERY_-----_
            "+query);
122         //Finding Node with best match using the directory
            received from the Boot strap server
123         //update arraylist to include query words
124         if (clusterNeighbours!=null)
125     {
126         Set keysIPS = clusterNeighbours.keySet();
127         Object [] keysArray = (Object []) keysIPS.toArray();
128         Directory dir=null;
129         for (int j=0;j<keysArray.length;j++)
130     {
131         ArrayList updated=new ArrayList();
132         dir = clusterNeighbours.get((String) keysArray[j]);
133         ArrayList a = dir.keywords;
134         System.out.println(a);
135         Matrix weights = dir.keyWeights;
136         double [][] weightsMatrix = weights.getArray();
137
138         for (int k=0;k<weights.getRowDimension();k++)
139     {
140         if (weightsMatrix[k][0]==0)
141         {
142         } else if (weightsMatrix[k][0]>=1)
143         {
144             for (int u=0;u<weightsMatrix[k][0];u++)
145         {
146             updated.add(a.get(k));
147         }
148         }
149     }

```



```

150
151         Message message = new Message((String)keysArray[j
                                ], updated);
152         database.add(message.getIP());
153         finder = new Finder(message);
154     }
155     //query
156     StringTokenizer st = new StringTokenizer(query);
157     ArrayList queryList = new ArrayList();
158     while(st.hasMoreTokens())
159     {
160         queryList.add((String)st.nextToken());
161     }
162     Message mQ = new Message("QUERY", queryList);
163     database.add(mQ.getIP());
164     finder = new Finder(mQ);
165     database.compute();
166     cont=" ";
167     query=" ";
168 } else
169 {
170     System.out.println("Cluster_Neighbours_got_issue_or_
                                the_Information_Agent_is_not_online_yet.");
171 }
172
173 Matrix simR=database.getSimMatrix();
174 double [][] simRArray = simR.getArray();
175 double mins = Double.parseDouble(minSup);
176 int indexer=0;
177 double simVal=0.0;
178 double temp=0.0;
179
180 for(int q=0;q<simRArray.length;q++)
181 {
182     simVal=simRArray[q][0];
183     if(simVal>mins && simVal>temp)
184     {
185         indexer=q;
186         temp=simVal;
187     }

```

```

188         }
189
190         ArrayList clientAgents=database.getClientList();
191         String chosen = (String)clientAgents.get(indexer);
192         System.out.println("THE_CHOSEN_ONE_IS_"+chosen);
193
194         ACLMessage reply = messagerec.createReply();
195         if(keywords!=null)
196         {
197             reply.setPerformative(ACLMessage.INFORM);
198             reply.setContent(chosen);
199         }
200         myAgent.send(reply);
201         System.out.println(keywords!=null ? "Informed_"+messagerec
            .getSender().getLocalName()+"_about_chosen_node_"+
            reply.getContent() : "Node_did_not_exist");
202     }else
203     {
204         block();
205     }
206 }catch(Exception e)
207 {
208     e.printStackTrace();
209 }
210 }
211 }//end inner class Node Requestor
212
213 private class Message
214 {
215     private String ip_Address;
216     private ArrayList keywords;
217
218     public Message(String ip_Address, ArrayList keywords)
219     {
220         this.ip_Address=ip_Address;;
221         this.keywords=keywords;
222     }
223
224     public void setIP(String ip_Address)
225     {

```

```

226         this.ip_Address=ip_Address;
227     }
228
229     public String getIP()
230     {
231         return ip_Address;
232     }
233
234     public void setKeywords(ArrayList keywords)
235     {
236         this.keywords=keywords;
237     }
238
239     public ArrayList getKeywords()
240     {
241         return keywords;
242     }
243
244 }//end inner class
245
246 //inner class to find node
247 private class Finder
248 {
249     private Message message;
250     private String tk[];
251     private int size;
252     private SortedMap map;
253
254     public Finder(Message message)
255     {
256         this.message=message;
257         map=new TreeMap();
258         getKeywords();
259     }
260
261     public void getKeywords()
262     {
263         int index=0;
264         String keywordSet=" ";
265         ArrayList keywords1 = message.getKeywords();

```

```

266         for (int i=0;i<keywords1.size();i++)
267         {
268             keywordSet+=(String)keywords1.get(i)+" ";
269         }
270         //Tokenize
271         StringTokenizer token = new StringTokenizer(keywordSet);
272         size=token.countTokens();
273         makeTKArray(size);
274         while(token.hasMoreTokens())
275         {
276             tk[index]=token.nextToken();
277             findTokenFrequency(tk[index]);
278         }
279         database.addKeywords(map);
280     }
281
282     public void makeTKArray(int size)
283     {
284         tk = new String[size];
285     }
286
287     public void findTokenFrequency(String token)
288     {
289         if(!map.containsKey(token))
290         {
291             map.put(token.toLowerCase(),1);
292         } else
293         {
294             Integer frequency = (Integer)map.get(token);
295             int freqVal = frequency.intValue();
296             freqVal+=1;
297             map.remove(token);
298             map.put(token, freqVal);
299         }
300     }
301 } //end inner class
302
303 private class Database
304 {
305     private ArrayList ipCols;

```

```

306     private ArrayList<SortedMap> tempMaster;
307     private double [][] masterKeywordMatrix;
308     private double [][] joiningNodeKeywords;
309     private ArrayList<String> listOfKeywords;
310     private Matrix U;
311     private Matrix S;
312     private Matrix V;
313     private Matrix S_inverse;
314     private Matrix V_transpose;
315     private Matrix Q_transpose;
316     private Matrix q;
317     private Matrix simM;
318     private String forIP="";
319
320     public Database()
321     {
322         ipCols = new ArrayList();
323         tempMaster = new ArrayList<SortedMap>();
324     }
325
326     public void add(String ip)
327     {
328         //ipCols.add(ip);
329         if (!ipCols.contains(ip))
330         {
331             ipCols.add(ip);
332             forIP=ip;
333         } else
334         {
335             int index=ipCols.indexOf(ip);
336             ipCols.remove(index);
337             tempMaster.remove(index);
338             ipCols.add(ip);
339             forIP=ip;
340         }
341     }
342
343     public void remove(String ip)
344     {
345         if (ipCols.contains(ip))

```

```

346         {
347             int index=ipCols.indexOf(ip);
348             ipCols.remove(index);
349             tempMaster.remove(index);
350         }
351     }
352
353     public void addKeywords(SortedMap map)
354     {
355         tempMaster.add(map);
356         prepareMatrix();
357     }
358
359     public void prepareMatrix()
360     {
361         SortedMap completeList = new TreeMap();
362         SortedMap temp = new TreeMap();
363         for (int i=0;i<tempMaster.size();i++)
364         {
365             temp=tempMaster.get(i);
366             Set keywords = temp.keySet();
367             Iterator itKeys = keywords.iterator();
368             while(itKeys.hasNext())
369             {
370                 String key = (String)itKeys.next();
371                 if(!completeList.containsKey(key))
372                 {
373                     completeList.put(key,0);
374                 }
375             }//end while
376         }//end loop
377
378         setCompleteList(completeList);
379
380         masterKeywordMatrix = new double[completeList.size()][ipCols.size()];
381         int rows = masterKeywordMatrix.length;
382         int cols = masterKeywordMatrix[0].length;
383         for (int n=0;n<cols;n++)
384         {

```

```

385         temp= new TreeMap();
386         Set completeKeys = completeList.keySet();
387         Iterator it = completeKeys.iterator();
388         temp=tempMaster.get(n);
389         for (int m=0;m<rows;m++)
390         {
391             while (it.hasNext())
392             {
393                 String key = (String)it.next();
394                 if (temp.containsKey(key))
395                 {
396                     int val = ((Integer)temp.get(key)).intValue();
397                     masterKeywordMatrix[m][n]=val;
398                 } else
399                 {
400                     masterKeywordMatrix[m][n]=0.0;
401                 }
402                 m++;
403             }
404         }
405     }
406 }
407
408 public void compute()
409 {
410     int rows = masterKeywordMatrix.length;
411     int cols = masterKeywordMatrix[0].length;
412     if (cols==1)
413     {
414         //do nothing
415         //to protect system from issuing dummy
416     } else
417     {
418         //display
419         System.out.println("Master_Matrix");
420         displayMatrix(masterKeywordMatrix,cols);
421         //calculate SVD considering the joining column ip address and
422         //its keywords is the new query(joining node keywords).
423         double[][] tempKeywordMatrix = new double[rows][cols-1];
424         joiningNodeKeywords=new double[rows][1];

```

```

424         int runner=0;
425         for ( int b=0;b<cols;b++)
426         {
427             for ( int a=0;a<rows;a++)
428             {
429                 if (b==ipCols.indexOf(forIP))
430                 {
431                     //do not get that column
432                     //make it joining node
433                     //joined keywords
434                     for ( int z=0;z<rows;z++)
435                     {
436                         joiningNodeKeywords[z][0]=masterKeywordMatrix[
437                             z][b];
438                     }
439                 } else
440                 {
441                     tempKeywordMatrix[a][runner]=masterKeywordMatrix[a
442                         ][b];
443                 }
444             }
445             runner++;
446         }
447
448         System.out.println("Compared Against Matrix");
449         displayMatrix(tempKeywordMatrix,tempKeywordMatrix[0].length);
450         System.out.println("Joining Matrix");
451         displayMatrix(joiningNodeKeywords,joiningNodeKeywords[0].
452             length);
453         Matrix Q = new Matrix(joiningNodeKeywords);
454         Q_transpose=Q.transpose();
455         calculateSVD(tempKeywordMatrix);
456         calculateq();
457         Matrix sim = calculateSim();
458         System.out.println("Similarity Report");
459         sim.print(sim.getColumnDimension(),3);
460     }

```

/**


```

461      * Displays matrix
462      */
463      public void displayMatrix(double[][] matrix,int cols)
464      {
465          Matrix mat = new Matrix(matrix);
466          mat.print(cols,1);
467      }
468
469      /**
470      * SVD calculation
471      */
472      public void calculateSVD(double[][] matrix)
473      {
474          Matrix mat = new Matrix(matrix);
475          SingularValueDecomposition svd = mat.svd();
476          U = svd.getU();
477          //U.print(ipCols.size(),3);
478          S = svd.getS();
479          //S.print(ipCols.size(),3);
480          S_inverse = S.inverse();
481          V = svd.getV();
482          //V.print(ipCols.size(),3);
483          V_transpose= V.transpose();
484      }
485
486      /**
487      * Computing query vector
488      */
489      public void calculateq()
490      {
491          q = (Q_transpose.times(U)).times(S_inverse);
492      }
493
494      public Matrix calculateSim()
495      {
496          double[][] vofQuery = q.toArray();
497          double[][] vofTerm = V_transpose.toArray();
498          double[][] sim=new double[vofQuery[0].length][1];
499          double[] num=new double[vofTerm.length];
500          double den1=0;

```

```

501         double [] den2=new double[vofTerm.length];
502
503         for (int x =0;x<vofTerm.length;x++)
504         {
505             for (int i =0;i<vofQuery[0].length;i++)
506             {
507                 num[x] +=vofQuery[0][i]*vofTerm[i][x];
508             }
509         }
510
511         for (int x=0;x<vofQuery[0].length;x++)
512         {
513             den1+=vofQuery[0][x]*vofQuery[0][x];
514         }
515
516         den1 = Math.sqrt(den1);
517
518         for (int i=0;i<den2.length;i++)
519         {
520             for (int x=0;x<vofTerm.length;x++)
521             {
522                 den2[i]+=vofTerm[x][i]*vofTerm[x][i];
523             }
524         }
525
526         for (int x=0;x<den2.length;x++)
527         {
528             den2[x] = Math.sqrt(den2[x]);
529         }
530
531         for (int i=0;i<sim.length;i++)
532         {
533             sim[i][0]=num[i]/(den1*den2[i]);
534         }
535
536         simM = new Matrix(sim);
537         return simM;
538     }
539
540     public Matrix getSimMatrix()

```

```
541     {
542         return simM;
543     }
544
545     public ArrayList getClientList ()
546     {
547         return ipCols;
548     }
549
550     public Matrix getMasterKeywordList ()
551     {
552         Matrix master = new Matrix(masterKeywordMatrix);
553         return master;
554     }
555
556     public void setCompleteList(SortedMap completeList)
557     {
558         Set keys = completeList.keySet();
559         listOfKeywords = new ArrayList<String>();
560         Iterator it = keys.iterator();
561         while(it.hasNext())
562         {
563             listOfKeywords.add((String)it.next());
564         }
565     }
566
567     public ArrayList getKeywordList ()
568     {
569         return listOfKeywords;
570     }
571 } //end inner class Message
572 } //end class
```

C.11 Class LocalAgent.java

```

1  import jade.core.*;
2  import java.util.*;
3  import jade.core.behaviours.*;
4  import jade.lang.acl.*;
5  import Jama.*;
6
7  /**
8   * Local Agent is an agent that holds information i.e.
9   * keys for defining local resources and the corresponding location of
10   * resource on the peer.
11   * @author M. Singh
12   * @version 1.3
13   */
14 public class LocalAgent extends Agent
15 {
16     private String hostaddress="";
17     private String name="";
18     private LocalUI ui;
19     private Hashtable<String,String> table = new Hashtable<String,String>();
20     private ArrayList<String> keywords = new ArrayList<String>();
21     private LocalDatabase database = new LocalDatabase();
22     private FrequencyFinder finder;
23     protected void setup()
24     {
25         //welcome
26         name = getAID().getName();
27         System.out.println("Hello_I_am_Local_Agent_and_my_name_is_"+name);
28
29         //instance of GUI
30         ui=new LocalUI();
31         ui.setAgent(this);
32         callAskUser();
33
34         //behaviour
35         addBehaviour(new CallForRegistration());
36
37         //behaviour

```

```

38         addBehaviour(new ServeIncomingMessage());
39     }
40
41     protected void takeDown()
42     {
43         ui.dispose();
44         System.out.println("Local Agent "+getAID().getName()+" Terminating");
45     }
46
47     public void callAskUser()
48     {
49         ui.askUser();
50     }
51
52     public void updateTable(Hashtable<String,String> catalog)
53     {
54         addBehaviour(new FileManager(this, catalog));
55     }
56
57     //inner class File Manager
58     private class FileManager extends TickerBehaviour
59     {
60         private FileManager(Agent a, Hashtable<String,String> catalog)
61         {
62             super(a,300000);
63             table=catalog;
64         }
65
66         public void onStart()
67         {
68             Set keys = table.keySet();
69             Iterator<String> it = keys.iterator();
70             while(it.hasNext())
71             {
72                 String key=it.next();
73                 String values=table.get(key);
74                 StringTokenizer st = new StringTokenizer(values);
75                 while(st.hasMoreTokens())
76                 {
77                     keywords.add(st.nextToken());

```

```

78         }
79     } //end while
80     System.out.println(keywords);
81 } //end onStart
82
83     public void onTick()
84     {
85         callAskUser();
86     }
87 } //end inner class File Manager
88
89 //inner class Call for Registration
90 private class CallForRegistration extends SimpleBehaviour
91 {
92     private MessageTemplate mt = MessageTemplate.and(MessageTemplate.
        MatchConversationId("keywords-request"), MessageTemplate.
        MatchPerformative(ACLMMessage.REQUEST));
93     public boolean done()
94     {
95         return false;
96     }
97
98     public void action()
99     {
100         try
101         {
102             ACLMessage message=myAgent.receive(mt);
103             if(message!=null)
104             {
105                 ui.informUser("Request_from_"+message.getSender().
                    getLocalName()+"\n"+"information_agent_to_issue_the_
                    keywords");
106                 ACLMessage reply = message.createReply();
107                 if(keywords!=null)
108                 {
109                     reply.setPerformative(ACLMMessage.INFORM);
110                     reply.setContentObject(keywords);
111                 }
112                 myAgent.send(reply);
113                 ui.informUser(keywords!=null ? "Informed_"+message.

```



```

149 String [] myCont = cont.split(":");
150 String query=myCont[0];
151 double minSup=Double.parseDouble(myCont[1]);
152 System.out.println("THE_QUERY_RECEIVED_BY_LOCAL_AGENT_
    "+query);
153
154 //all keywords for all documents are stored in hash
    table -> table as (filename->keywords) as key->
    value pairs
155 Set keys=table.keySet();
156 Object [] key=(Object []) keys.toArray();
157 for (int i=0;i<key.length;i++)
158 {
159     ArrayList keySet = new ArrayList();
160     String tempKey = (String)table.get(key[i]);
161     StringTokenizer st = new StringTokenizer(tempKey);
162     while(st.hasMoreTokens())
163     {
164         keySet.add((String)st.nextToken());
165     }
166     Transport message = new Transport((String)key[i],
        keySet);
167     database.add(message.getIP());
168     finder = new FrequencyFinder(message);
169 }
170 //query
171 StringTokenizer st = new StringTokenizer(query);
172 ArrayList queryList = new ArrayList();
173 while(st.hasMoreTokens())
174 {
175     queryList.add((String)st.nextToken());
176 }
177 Transport mQ = new Transport("QUERY", queryList);
178 database.add(mQ.getIP());
179 finder = new FrequencyFinder(mQ);
180 database.compute();
181
182 //prepare reply
183 Matrix simR=database.getSimMatrix();
184 double [][] simRArray = simR.getArray();

```



```

185         int indexer=0;
186         double simVal=0.0;
187         double temp=0.0;
188
189         for ( int q=0;q<simRArray.length;q++)
190         {
191             simVal=simRArray[q][0];
192             if (simVal>minSup && simVal>temp)
193             {
194                 indexer=q;
195                 temp=simVal;
196             }
197         }
198
199         ArrayList docs=database.getClientList();
200         String chosen = (String)docs.get(indexer);
201         System.out.println("THE_CHOSEN_DOCUMENT_IS_"+chosen);
202         MatchStore matchStore = new MatchStore(chosen,
203             getLocalName(),simVal);
204
205         //reply
206         ACLMessage reply = request.createReply();
207         if (chosen!=null)
208         {
209             reply.setPerformative(ACLMessage.INFORM);
210             reply.setContentObject(matchStore);
211         }
212         myAgent.send(reply);
213         System.out.println(keywords!=null ? "Informed_"+
214             request.getSender().getLocalName()+"_about_chosen_
215             document_" : "Document_did_not_exist");
216
217     } else
218     {
219         System.out.println("No_message_yet");
220         block();
221     }
222 } //end while
223 } catch (Exception e)
224 {
225     e.printStackTrace();

```

```
222         }
223     }
224 } //end inner class serve incoming message
225
226 private class Transport
227 {
228     private String ip_Address;
229     private ArrayList keywords;
230
231     public Transport(String ip_Address, ArrayList keywords)
232     {
233         this.ip_Address=ip_Address;;
234         this.keywords=keywords;
235     }
236
237     public void setIP(String ip_Address)
238     {
239         this.ip_Address=ip_Address;
240     }
241
242     public String getIP()
243     {
244         return ip_Address;
245     }
246
247     public void setKeywords(ArrayList keywords)
248     {
249         this.keywords=keywords;
250     }
251
252     public ArrayList getKeywords()
253     {
254         return keywords;
255     }
256
257 } //end inner class
258
259 //inner class to find resource
260 private class FrequencyFinder
261 {
```

```

262     private Transport message;
263     private String tk[];
264     private int size;
265     private SortedMap map;
266
267     public FrequencyFinder(Transport message)
268     {
269         this.message=message;
270         map=new TreeMap();
271         getKeywords();
272     }
273
274     public void getKeywords()
275     {
276         int index=0;
277         String keywordSet="";
278         ArrayList keywords1 = message.getKeywords();
279         for (int i=0;i<keywords1.size();i++)
280         {
281             keywordSet+=(String)keywords1.get(i)+"_";
282         }
283         //Tokenize
284         StringTokenizer token = new StringTokenizer(keywordSet);
285         size=token.countTokens();
286         makeTKArray(size);
287         while(token.hasMoreTokens())
288         {
289             tk[index]=token.nextToken();
290             findTokenFrequency(tk[index]);
291         }
292         database.addKeywords(map);
293     }
294
295     public void makeTKArray(int size)
296     {
297         tk = new String[size];
298     }
299
300     public void findTokenFrequency(String token)
301     {

```

```

302         if (!map.containsKey(token))
303         {
304             map.put(token.toLowerCase(), 1);
305         } else
306         {
307             Integer frequency = (Integer)map.get(token);
308             int freqVal = frequency.intValue();
309             freqVal+=1;
310             map.remove(token);
311             map.put(token, freqVal);
312         }
313     }
314 } //end inner class
315
316 private class LocalDatabase
317 {
318     private ArrayList ipCols;
319     private ArrayList<SortedMap> tempMaster;
320     private double[][] masterKeywordMatrix;
321     private double[][] joiningNodeKeywords;
322     private ArrayList<String> listOfKeywords;
323     private Matrix U;
324     private Matrix S;
325     private Matrix V;
326     private Matrix S_inverse;
327     private Matrix V_transpose;
328     private Matrix Q_transpose;
329     private Matrix q;
330     private Matrix simM;
331     private String forIP="";
332
333     public LocalDatabase()
334     {
335         ipCols = new ArrayList();
336         tempMaster = new ArrayList<SortedMap>();
337     }
338
339     public void add(String ip)
340     {
341         //ipCols.add(ip);

```

```

342         if (!ipCols.contains(ip))
343         {
344             ipCols.add(ip);
345             forIP=ip;
346         } else
347         {
348             int index=ipCols.indexOf(ip);
349             ipCols.remove(index);
350             tempMaster.remove(index);
351             ipCols.add(ip);
352             forIP=ip;
353         }
354     }
355
356     public void remove(String ip)
357     {
358         if (ipCols.contains(ip))
359         {
360             int index=ipCols.indexOf(ip);
361             ipCols.remove(index);
362             tempMaster.remove(index);
363         }
364     }
365
366     public void addKeywords(SortedMap map)
367     {
368         tempMaster.add(map);
369         prepareMatrix();
370     }
371
372     public void prepareMatrix()
373     {
374         SortedMap completeList = new TreeMap();
375         SortedMap temp = new TreeMap();
376         for (int i=0;i<tempMaster.size();i++)
377         {
378             temp=tempMaster.get(i);
379             Set keywords = temp.keySet();
380             Iterator itKeys = keywords.iterator();
381             while(itKeys.hasNext())

```

```

382         {
383             String key = (String) itKeys.next();
384             if (!completeList.containsKey(key))
385             {
386                 completeList.put(key, 0);
387             }
388         } //end while
389     } //end loop
390
391     setCompleteList ( completeList );
392
393     masterKeywordMatrix = new double [ completeList.size() ][ ipCols.size
394         () ];
395     int rows = masterKeywordMatrix.length;
396     int cols = masterKeywordMatrix[0].length;
397     for ( int n=0; n<cols; n++)
398     {
399         temp= new TreeMap();
400         Set completeKeys = completeList.keySet();
401         Iterator it = completeKeys.iterator();
402         temp=tempMaster.get(n);
403         for ( int m=0; m<rows; m++)
404         {
405             while ( it.hasNext() )
406             {
407                 String key = (String) it.next();
408                 if (temp.containsKey(key))
409                 {
410                     int val = ((Integer) temp.get(key)).intValue();
411                     masterKeywordMatrix[m][n]=val;
412                 } else
413                 {
414                     masterKeywordMatrix[m][n]=0.0;
415                 }
416                 m++;
417             }
418         }
419     }
420

```

```

421     public void compute()
422     {
423         int rows = masterKeywordMatrix.length;
424         int cols = masterKeywordMatrix[0].length;
425         if (cols==1)
426         {
427             //do nothing
428             //to protect system from issuing dummy
429         } else
430         {
431             //display
432             System.out.println("Master_Matrix");
433             displayMatrix(masterKeywordMatrix, cols);
434             //calculate SVD considering the joining column ip address and
435             //its keywords is the new query(joining node keywords).
436             double[][] tempKeywordMatrix = new double[rows][cols-1];
437             joiningNodeKeywords=new double[rows][1];
438             int runner=0;
439             for (int b=0;b<cols;b++)
440             {
441                 for (int a=0;a<rows;a++)
442                 {
443                     if (b==ipCols.indexOf(forIP))
444                     {
445                         //do not get that column
446                         //make it joining node
447                         //joined keywords
448                         for (int z=0;z<rows;z++)
449                         {
450                             joiningNodeKeywords[z][0]=masterKeywordMatrix[
451                                 z][b];
452                         }
453                     } else
454                     {
455                         tempKeywordMatrix[a][runner]=masterKeywordMatrix[a
456                             ][b];
457                     }
458                 }
459                 runner++;
460             }
461         }
462     }

```

```

458
459         System.out.println("Compared_Against_Matrix");
460         displayMatrix(tempKeywordMatrix,tempKeywordMatrix[0].length);
461         System.out.println("Joining_Matrix");
462         displayMatrix(joiningNodeKeywords,joiningNodeKeywords[0].
            length);
463         Matrix Q = new Matrix(joiningNodeKeywords);
464         Q_transpose=Q.transpose();
465         calculateSVD(tempKeywordMatrix);
466         calculateq();
467         Matrix sim = calculateSim();
468         System.out.println("Similarity_Report");
469         sim.print(sim.getColumnDimension(),3);
470     }
471 }
472
473 /**
474  * Displays matrix
475  */
476 public void displayMatrix(double[][] matrix,int cols)
477 {
478     Matrix mat = new Matrix(matrix);
479     mat.print(cols,1);
480 }
481
482 /**
483  * SVD calculation
484  */
485 public void calculateSVD(double[][] matrix)
486 {
487     Matrix mat = new Matrix(matrix);
488     SingularValueDecomposition svd = mat.svd();
489     U = svd.getU();
490     //U.print(ipCols.size(),3);
491     S = svd.getS();
492     //S.print(ipCols.size(),3);
493     S_inverse = S.inverse();
494     V = svd.getV();
495     //V.print(ipCols.size(),3);
496     V_transpose= V.transpose();

```



```

497     }
498
499     /**
500     * Computing query vector
501     */
502     public void calculateq()
503     {
504         q = (Q_transpose.times(U)).times(S_inverse);
505     }
506
507     public Matrix calculateSim()
508     {
509         double [][] vofQuery = q.toArray();
510         double [][] vofTerm = V_transpose.toArray();
511         double [][] sim=new double[vofQuery[0].length][1];
512         double [] num=new double[vofTerm.length];
513         double den1=0;
514         double [] den2=new double[vofTerm.length];
515
516         for (int x =0;x<vofTerm.length;x++)
517         {
518             for (int i =0;i<vofQuery[0].length;i++)
519             {
520                 num[x] +=vofQuery[0][i]*vofTerm[i][x];
521             }
522         }
523         System.out.println("NUMERATOR_"+num[0]);
524         //-----
525         for (int x=0;x<vofQuery[0].length;x++)
526         {
527             den1+=vofQuery[0][x]*vofQuery[0][x];
528         }
529
530         den1 = Math.sqrt(den1);
531         System.out.println("DENOMINATOR_PART_1_"+den1);
532
533         for (int i=0;i<den2.length;i++)
534         {
535             for (int x=0;x<vofTerm.length;x++)
536             {

```

```

537         den2[i] += vofTerm[x][i] * vofTerm[x][i];
538     }
539 }
540
541     for (int x=0; x<den2.length; x++)
542     {
543         den2[x] = Math.sqrt(den2[x]);
544         System.out.println("DENOMINATOR_PART_2_" + den2[x]);
545     }
546
547     for (int i=0; i<sim.length; i++)
548     {
549         sim[i][0] = num[i] / (den1 * den2[i]);
550         System.out.println("SIMILARITY_CALC_" + sim[i][0]);
551     }
552
553     simM = new Matrix(sim);
554     return simM;
555 }
556
557 public Matrix getSimMatrix()
558 {
559     return simM;
560 }
561
562 public ArrayList getClientList()
563 {
564     return ipCols;
565 }
566
567 public Matrix getMasterKeywordList()
568 {
569     Matrix master = new Matrix(masterKeywordMatrix);
570     return master;
571 }
572
573 public void setCompleteList(SortedMap completeList)
574 {
575     Set keys = completeList.keySet();
576     listOfKeywords = new ArrayList<String>();

```

```
577         Iterator it = keys.iterator();
578         while(it.hasNext())
579         {
580             listOfKeywords.add((String)it.next());
581         }
582     }
583
584     public ArrayList getKeywordList()
585     {
586         return listOfKeywords;
587     }
588 } //end inner class
589
590 } //end class
```

C.12 Class LocalUI.java

```

1  import java.io.*;
2  import java.util.*;
3  import javax.swing.*;
4  import jade.core.*;
5  /**
6   * User Interface for Local Agent
7   *
8   * @author M. Singh
9   * @version 1.1
10  */
11 public class LocalUI extends JFrame
12 {
13     private Hashtable<String,String> catalog;
14     private LocalAgent myAgent;
15     public LocalUI()
16     {
17         catalog=new Hashtable<String,String>();
18     }
19
20     public void setAgent(LocalAgent agent)
21     {
22         myAgent=agent;
23     }
24
25     public void askUser()
26     {
27         String option = JOptionPane.showInputDialog("Please_enter_YES/NO_for_
                updating_the_catalog");
28         if(option.toLowerCase().equals("yes"))
29         {
30             try
31             {
32                 File folder = new File("Shared");
33                 File[] listOfFiles = folder.listFiles();
34
35                 for(int i=0;i<listOfFiles.length;i++)
36                 {
37                     if(listOfFiles[i].isFile())

```

```
38         {
39             System.out.println("File_"+listOfFiles[i].getName());
40             String keywords = JOptionPane.showInputDialog("Please_
                    enter_the_keywords_describing_file_-_"+listOfFiles
                    [i].getName()+"\n"+"and_separate_using_space.");
41             catalog.put(listOfFiles[i].getName(), keywords.
                    toLowerCase());
42         }
43     }
44     } catch (Exception e)
45     {
46         e.printStackTrace();
47     }
48     myAgent.updateTable(catalog);
49 }
50 }
51
52 public void informUser(String message)
53 {
54     JOptionPane.showMessageDialog(null, message);
55     //System.out.println(message);
56 }
57
58 } //end class
```

C.13 Class MatchStore.java

```
1  import java.io.*;
2  import java.util.*;
3  /**
4   * Serialised Data Structure
5   *
6   * @author M. Singh
7   * @version 1.2
8   */
9  public class MatchStore implements Serializable
10 {
11     String chosenDocs;
12     String nameLA;
13     double similarityValues;
14
15     public MatchStore(String chosenDocs, String nameLA, double
        similarityValues)
16     {
17         this.chosenDocs=chosenDocs;
18         this.nameLA=nameLA;
19         this.similarityValues=similarityValues;
20     }
21 }
```

C.14 Class InterfaceAgent.java

```

1  import jade.core.*;
2  import jade.core.behaviours.*;
3  import jade.lang.acl.*;
4  import jade.gui.*;
5  import jade.content.*;
6  import jade.content.onto.basic.*;
7  import jade.content.lang.*;
8  import jade.content.lang.sl.*;
9  import jade.domain.*;
10 import jade.domain.mobility.*;
11 import jade.domain.JADEAgentManagement.*;
12 import java.util.*;
13
14 /**
15  * Interface Agent is an agent that provides user interaction to the system.
16  *
17  * @author M. Singh
18  * @version 1.7
19  */
20 public class InterfaceAgent extends GuiAgent
21 {
22     private String name="";
23     private SearchGUI gui;
24     jade.core.Runtime runtime=jade.core.Runtime.instance();
25     private jade.wrapper.AgentContainer home;
26     private int command;
27     private int count=(int)(Math.random()*100)+3000;
28     Vector agents=new Vector();
29
30     public static final int QUIT=0;
31     public static final int NEW_RECON_AGENT=1;
32     public static final int KILL_AGENT=4;
33
34     protected void setup()
35     {
36         //welcome
37         name=getAID().getName();
38         System.out.println("Interface_Agent_"+name+"_started.");

```

```

39
40     //register language and ontology
41     getContentManager().registerLanguage(new SLCodec());
42     getContentManager().registerOntology(MobilityOntology.getInstance());
43
44     //create agent container
45     home = runtime.createAgentContainer(new ProfileImpl());
46     doWait(2000);
47
48     //start gui
49     gui=new SearchGUI();
50     gui.setAgent(this);
51     gui.show();
52
53     addBehaviour(new ReceiveMessageRecon());
54
55     addBehaviour(new ReceiveTerminationRecon());
56 }
57
58 protected void onGuiEvent(GuiEvent ev)
59 {
60     command=ev.getType();
61     if(command==QUIT)
62     {
63         try
64         {
65             home.kill();
66         } catch (Exception e)
67         {
68             e.printStackTrace();
69         }
70         gui.setVisible(false);
71         gui.dispose();
72         doDelete();
73         System.exit(0);
74     }
75     if(command==NEW_RECON_AGENT)
76     {
77         jade.wrapper.AgentController a = null;
78         System.out.println("MUST_BE_CREATED");

```



```

79         try
80         {
81             Object [] args=new Object [5];
82             args[0]=getAID ();
83             System.out.println ( args [0] );
84             args[1]=gui.getQuery (); //query
85             args[2]="0.0"; //minimum support
86             args[3]=( Object )name;
87             args[4]="2"; //number of hops
88             String name_of_Agent="Reconnaissance_Agent_"+(count++);
89             a=home.createNewAgent ( name_of_Agent , ReconnaissanceAgent.class ,
                getName () , args );
90             a.start ();
91             agents.add ( name_of_Agent );
92             gui.activeAgents ( agents );
93         } catch (Exception ee)
94         {
95             System.out.println ( "Problem_while_creating_new_agent_"+ee );
96         }
97         return ;
98     }
99 }
100
101 protected void takeDown()
102 {
103     if ( gui!=null )
104     {
105         gui.setVisible ( false );
106         gui.dispose ();
107     }
108
109     System.out.println ( "Interface_terminating_for_"+name+"\n"+"Thank_you_
        for_using_AFFINITY." );
110     System.exit (0);
111 }
112
113 //inner class
114 private class ReceiveMessageRecon extends CyclicBehaviour
115 {
116     MatchStore matchStore=null;

```

```

117     MessageTemplate mt = MessageTemplate.and(MessageTemplate.
        MatchConversationId("results"), MessageTemplate.MatchPerformative(
            ACLMessage.INFORM));
118     public void action()
119     {
120         try
121         {
122             ACLMessage reply = receive(mt);
123             if(reply != null)
124             {
125                 matchStore=(MatchStore)reply.getContentObject();
126                 gui.setResult(matchStore);
127             } else
128             {
129                 block();
130             }
131         } catch(Exception e)
132         {
133             e.printStackTrace();
134         }
135     }
136 }
137
138 //inner class
139 private class ReceiveTerminationRecon extends SimpleBehaviour
140 {
141     private boolean check=false;
142     MessageTemplate mt = MessageTemplate.and(MessageTemplate.
        MatchConversationId("termination-instruction"), MessageTemplate.
        MatchPerformative(ACLMessage.INFORM));
143
144     public boolean done()
145     {
146         return check;
147     }
148
149     public void action()
150     {
151         try
152         {

```

```
153         ACLMessage message = receive(mt);
154         if (message != null)
155             {
156                 agents.remove(message.getSender().getLocalName());
157                 gui.activeAgents(agents);
158                 check=true;
159             } else
160             {
161                 block();
162             }
163         } catch (Exception e)
164         {
165             e.printStackTrace();
166         }
167     }
168 }
169 } //end class
```

C.15 Class SearchGUI.java

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.border.*;
5  import javax.swing.event.*;
6  import jade.core.*;
7  import java.util.*;
8  import jade.gui.*;
9
10 /**
11  * SearchGUI is User interface for Interface Agent
12  *
13  */
14 public class SearchGUI extends JFrame
15 {
16     private InterfaceAgent myAgent;
17     private String query="";
18     // Variables declaration
19     private JLabel jLabel2;
20     private JLabel jLabel3;
21     private JTextArea jTextArea1;
22     private JScrollPane jScrollPane3;
23     private JList jList1;
24     private DefaultListModel listModel1;
25     private JScrollPane jScrollPane2;
26     private JTabbedPane jTabbedPane1;
27     private JPanel contentPane;
28     //-----
29     private JLabel jLabel1;
30     private JTextField jTextField1;
31     private JButton jButton1;
32     private JPanel jPanel1;
33     //-----
34     // End of variables declaration
35
36     public void setAgent(InterfaceAgent a)
37     {
38         myAgent=a;

```

```

39         setTitle("Affinity - Search Node - "+myAgent.getName());
40     }
41
42     public SearchGUI()
43     {
44         super();
45         JFrame.setDefaultLookAndFeelDecorated(true);
46         JDialog.setDefaultLookAndFeelDecorated(true);
47         try
48         {
49             UIManager.setLookAndFeel("com.sun.java.swing.plaf.
                    windows.WindowsLookAndFeel");
50         }
51         catch (Exception ex)
52         {
53             System.out.println("Failed loading L&F: ");
54             System.out.println(ex);
55         }
56         addWindowListener(new WindowAdapter() {
57             public void windowClosing(WindowEvent e) {
58                 myAgent.doDelete();
59             }
60         });
61         initComponents();
62     }
63
64     private void initComponents()
65     {
66         jLabel2 = new JLabel();
67         jLabel3 = new JLabel();
68         jTextArea1 = new JTextArea();
69         jScrollPane3 = new JScrollPane();
70         listModel1 = new DefaultListModel();
71         jList1 = new JList(listModel1);
72         jScrollPane2 = new JScrollPane();
73         jTabbedPane1 = new JTabbedPane();
74         contentPane = (JPanel) this.getContentPane();
75         //-----
76         jLabel1 = new JLabel();
77         jTextField1 = new JTextField();

```

```

78         jButton1 = new JButton();
79         jPanel1 = new JPanel();
80         //————
81
82         //
83         // jLabel2
84         //
85         jLabel2.setText("Search_Results_for_Query:_");
86         //
87         // jLabel2
88         //
89         jLabel3.setText("Active_Reconnaissance_Agents");
90         //
91         // JTextArea1
92         //
93         JTextArea1.setFont(new java.awt.Font("Tahoma", 0, 11));
94         JTextArea1.setToolTipText("Search_Results");
95         JTextArea1.setEditable(false);
96         JTextArea1.setLineWrap(true);
97         //
98         // JScrollPane3
99         //
100        JScrollPane3.setViewportView(JTextArea1);
101        //
102        // JList1
103        //
104        JList1.setVisibleRowCount(7);
105        JList1.setFixedCellHeight(18);
106        JList1.setSelectionMode(ListSelectionModel.
            SINGLE_INTERVAL_SELECTION);
107        //
108        // JScrollPane2
109        //
110        JScrollPane2.setViewportView(JList1);
111        //
112        // JTabbedPane1
113        //
114        JTabbedPane1.addTab("Search", jPanel1);
115        JTabbedPane1.setBackground(new Color(255, 255, 255));
116        JTabbedPane1.addChangeListener(new ChangeListener() {

```

```
117         public void stateChanged(ChangeEvent e)
118         {
119             jTabbedPane1_stateChanged(e);
120         }
121
122     });
123     //
124     // contentPane
125     //
126     contentPane.setLayout(null);
127     contentPane.setBorder(BorderFactory.createRaisedBevelBorder());
128     ;
129     addComponent(contentPane, jLabel2, 211,15,337,18);
130     addComponent(contentPane, jLabel3, 550,15,157,18);
131     addComponent(contentPane, jScrollPane3, 210,33,337,328);
132     addComponent(contentPane, jScrollPane2, 550,33,157,100);
133     addComponent(contentPane, jTabbedPane1, 4,11,200,350);
134     //
135     // jLabel1
136     //
137     jLabel1.setText("Enter_Search_Query");
138     //
139     // jButton1
140     //
141     jButton1.setText("Search");
142     jButton1.setToolTipText("Click_to_Start_Search");
143     jButton1.addActionListener(new ActionListener() {
144         public void actionPerformed(ActionEvent e)
145         {
146             jButton1_actionPerformed(e);
147         }
148     });
149     //
150     // jPanel1
151     //
152     jPanel1.setLayout(null);
153     jPanel1.setBorder(new TitledBorder("Search_Query_Window"));
154     jPanel1.setBackground(new Color(255, 254, 254));
155     jPanel1.setOpaque(false);
```

```

156         jPanel1.setToolTipText("Search");
157         addComponent(jPanel1, jLabel1, 5, 50, 100, 18);
158         addComponent(jPanel1, jTextField1, 5, 70, 180, 22);
159         addComponent(jPanel1, jButton1, 48, 92, 83, 28);
160         //
161         // SearchGUI
162         //
163         this.setLocation(new Point(0, 0));
164         this.setSize(new Dimension(730, 400));
165         this.setResizable(false);
166     }
167
168     /** Add Component Without a Layout Manager (Absolute Positioning) */
169     private void addComponent(Container container, Component c, int x, int y,
170                               int width, int height)
171     {
172         c.setBounds(x, y, width, height);
173         container.add(c);
174     }
175
176     private void jTabbedPane1_stateChanged(ChangeEvent e)
177     {
178         System.out.println("NOTHING_SHOULD_BE_HAPPENING_HERE");
179     }
180
181     private void jButton1_actionPerformed(ActionEvent e)
182     {
183         query = jTextField1.getText();
184         jTextField1.setText("");
185         jLabel2.setText("Search_Results_for_Query:_"+query);
186         GuiEvent ge = new GuiEvent(this, myAgent.NEW_RECON_AGENT);
187         myAgent.postGuiEvent(ge);
188     }
189
190     public Object getQuery()
191     {
192         return (Object) query;
193     }
194
195     public void activeAgents(Vector agents)

```



```
195     {
196         listModel1.clear();
197         for (int i=0;i<agents.size();i++)
198         {
199             listModel1.addElement(agents.get(i));
200         }
201     }
202
203     public void setResult(MatchStore matchStore)
204     {
205         jTextArea1.append("Manu_Manu_Manu\n");
206         jTextArea1.append(matchStore.nameLA+"\n");
207         jTextArea1.append(matchStore.chosenDocs+"\n");
208         jTextArea1.append(""+matchStore.similarityValues+"\n");
209         jTextArea1.append("\n");
210     }
211 } //end class
```

C.16 Class ReconnaissanceAgent.java

```

1  import jade.core.*;
2  import jade.core.behaviours.*;
3  import jade.lang.acl.*;
4  import jade.domain.*;
5  import jade.domain.mobility.*;
6  import jade.domain.JADEAgentManagement.WhereIsAgentAction;
7  import jade.domain.JADEAgentManagement.KillAgent;
8  import jade.content.*;
9  import jade.content.onto.basic.*;
10 import jade.content.lang.*;
11 import jade.content.lang.sl.*;
12 import java.util.*;
13 /**
14  * ReconnaissanceAgent is a mobile agent that is created by the Interface
15  *   Agent
16  *
17  * @author M. Singh
18  * @version 2.5
19  */
20 public class ReconnaissanceAgent extends Agent
21 {
22     private String creator="";
23     private String query="";
24     private String minSup="";
25     private int maxHops;
26     private String nameIA="";
27     private String cont="";
28     private Map locations=new HashMap();
29     private String destName="";
30     private String destLAName="";
31     private int hopNumber=0;
32
33     protected void setup()
34     {
35         //register language and ontology
36         getContentManager().registerLanguage(new SLCodec());
37         getContentManager().registerOntology(MobilityOntology.getInstance());

```

```

38
39     System.out.println("Hi, _I_am_Reconnaissance_Agent__"+getLocalName());
40
41     //get arguments passed while creation of reconnaissance agent
42     Object[] args = getArguments();
43     creator=(String) args[3];
44     if(creator.startsWith("S"))
45     {
46         nameIA=creator;
47         nameIA=nameIA.replace("S", "I");
48     }
49     query=(String) args[1];
50     minSup=(String) args[2];
51     cont=query+": "+minSup;
52     maxHops=Integer.parseInt((String) args[4]);
53
54     //request location
55     String nameofAgent=getNode(nameIA);
56     destLAName=nameofAgent;
57     System.out.println(nameofAgent);
58     commForJump(nameofAgent);
59
60 }
61
62 protected void takeDown()
63 {
64     System.out.println("Terminating_Myself");
65 }
66
67 protected void afterMove()
68 {
69     //register language and ontology
70     getContentManager().registerLanguage(new SLCodec());
71     getContentManager().registerOntology(MobilityOntology.getInstance());
72
73     hopNumber++;
74     //1. recon agent has to find local agent and compare the query against
       the catalog it is keeping
75     //if any of the results are good using MinSup it Sends ACL Message to
       creator (Interface Agent)

```

```

76      //informing about the find (possible name of file and its name of
       local agent keep it.
77
78      //Send Message to LA
79      System.out.println("THE_DESTINATION_LOCAL_AGENT_IS_"+destLAName);
80      ACLMessage request=new ACLMessage(ACLMessage.REQUEST);
81      request.addReceiver(new AID(destLAName,AID.ISGUID));
82      request.setConversationId("search-request");
83      request.setReplyWith("request"+System.currentTimeMillis());
84      request.setContent(cont);
85      send(request);
86      System.out.println("Message_sent_to_"+destLAName);
87
88      //Prepare message receiving template from LA about the matches found
89      MessageTemplate mt = MessageTemplate.and(MessageTemplate.
          MatchConversationId("search-request"),MessageTemplate.
          MatchInReplyTo(request.getReplyWith()));
90      ACLMessage reply = blockingReceive(mt);
91      MatchStore matchStore=null;
92      if(reply!=null)
93      {
94          if(reply.getPerformative()==ACLMessage.INFORM)
95          {
96              try
97              {
98                  matchStore = (MatchStore)reply.getContentObject();
99              }catch(Exception e)
100              {
101                  e.printStackTrace();
102              }
103          }
104      }
105
106      //prepare to send message to interface agent (home) about the matches
       found
107      try
108      {
109          System.out.println("Sending_Message_to_home");
110          ACLMessage inform = new ACLMessage(ACLMessage.INFORM);
111          inform.addReceiver(new AID(creator,AID.ISGUID));

```

```

112         inform.setConversationId("results");
113         inform.setContentObject(matchStore);
114         send(inform);
115     } catch (Exception e)
116     {
117         e.printStackTrace();
118     }
119
120     //2. recon agent checks if it has made number of jumps less than
121     //maximum number of hops allowed.
122     //if it is less then it communicate to information agent here on this
123     //node and get the next jump
124     //address and conatiner
125     //else it kills itself.
126     if (hopNumber < maxHops)
127     {
128         System.out.println("TIME_TO_JUMP_TO_NEXT_DESTINATION");
129         String nameI = "";
130         //request location
131         if (creator.startsWith("S"))
132         {
133             nameI = destLAName;
134             nameI = nameI.replace("L", "I");
135         }
136         String nameofAgent = getNode(nameI);
137         destLAName = nameofAgent;
138         String creatorLA = "";
139         if (creator.startsWith("S"))
140         {
141             creatorLA = creator;
142             creatorLA = creatorLA.replace("S", "L");
143         }
144         if (!destLAName.equals(creatorLA))
145         {
146             System.out.println("NEXT_JUMP_IS_TOWARDS_CONTAINER_CONTAINING_
147                                AGENT_NAME: - " + nameofAgent);
148             //jumping time
149             commForJump(nameofAgent);
150         } else
151         {

```

```

149         System.out.println("NO_SUITABLE_NODES_FOUND");
150         doDelete();
151     }
152 } else
153 {
154     //Preparing to die
155     ACLMessage message = new ACLMessage(ACLMessage.INFORM);
156     message.addReceiver(new AID(creator, AID.ISGUID));
157     message.setConversationId("termination-instruction");
158     send(message);
159     //time to die
160
161     System.out.println("Terminating_Myself");
162
163     /*
164     KillAgent ka=new KillAgent();
165     ka.setAgent(getAID());
166     sendRequest(new Action(getAID(),ka));
167     */
168     doWait(3000);
169     doDelete();
170 }
171 }
172
173
174 public void commForJump(String nameofAgent)
175 {
176     try
177     {
178         AID aid = new AID(nameofAgent, AID.ISGUID);
179         WhereIsAgentAction where = new WhereIsAgentAction();
180         where.setAgentIdentifier(aid);
181         //send message to AMS
182         sendRequest(new Action(getAMS(), where));
183
184         //receiving message from AMS
185         MessageTemplate mt = MessageTemplate.and(MessageTemplate.
            MatchSender(getAMS()), MessageTemplate.MatchPerformative(
            ACLMessage.INFORM));
186         ACLMessage resp = blockingReceive(mt);

```

```

187         ContentElement ce = getContentManager().extractContent(resp);
188         Result result = (Result)ce;
189         jade.util.leap.Iterator it = result.getItems().iterator();
190         while(it.hasNext())
191         {
192             Location loc=(Location)it.next();
193             locations.put(loc.getName(),loc);
194             destName=loc.getName();
195         }
196
197         doWait(5000);
198         System.out.println("Wait_Finished");
199         //name of agent to be tranfered that is reconnaissance agent
           itself
200         AID aidi = new AID(getLocalName(),AID.ISLOCALNAME);
201         Location dest = (Location)locations.get(destName);
202         MobileAgentDescription mad = new MobileAgentDescription();
203         mad.setName(aid);
204         mad.setDestination(dest);
205         MoveAction ma = new MoveAction();
206         ma.setMobileAgentDescription(mad);
207         sendRequest(new Action(aid,ma));
208         doMove(dest);
209         System.out.println("SHOULD_HAVE_MOVED_BY_NOW");
210     } catch (Exception e)
211     {
212         e.printStackTrace();
213     }
214 }
215
216 //get node
217 public String getNode(String agentName)
218 {
219     String nodeName="";
220
221     //Send Message to IA
222     ACLMessage request=new ACLMessage(ACLMessage.REQUEST);
223     request.addReceiver(new AID(agentName,AID.ISGUID));
224     request.setConversationId("node-request");
225     request.setReplyWith("request"+System.currentTimeMillis());

```

```

226         request.setContent( cont );
227         send( request );
228
229         //Prepare message receiving template
230         MessageTemplate mt = MessageTemplate.and( MessageTemplate.
                MatchConversationId( "node-request" ), MessageTemplate.MatchInReplyTo
                ( request.getReplyWith() ) );
231         ACLMessage reply = blockingReceive( mt );
232
233         if( reply != null )
234         {
235             if( reply.getPerformative() == ACLMessage.INFORM )
236             {
237                 try
238                 {
239                     nodeName = (String) reply.getContent();
240                 } catch( Exception e )
241                 {
242                     e.printStackTrace();
243                 }
244             }
245         }
246
247         return nodeName;
248     } //end get Node
249
250     //send message to AMS for location of the named static agent
251     public void sendRequest( Action action )
252     {
253         ACLMessage request = new ACLMessage( ACLMessage.REQUEST );
254         request.setLanguage( new SLCodec().getName() );
255         request.setOntology( MobilityOntology.getInstance().getName() );
256         try
257         {
258             getContentManager().fillContent( request, action );
259             request.addReceiver( action.getActor() );
260             send( request );
261         } catch( Exception e )
262         {
263             e.printStackTrace();

```



```
264         }  
265     }  
266  
267 } //end class
```